VŠB – Technická univerzita Ostrava

Fakulta elektrotechniky a informatiky

Katedra informatiky

Informační systém správy vozového parku firmy BKB Metal a.s.

Information System for Managing Company Cars and Work Travels for the Company BKB Metal a.s.

2010 Bc. Ondřej Krhut

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne 7. května 2010 ……

Bc. Ondřej KRHUT

Poděkování

Rád bych na tomto místě poděkoval vedoucímu Ing. Pavlu Krömerovi za odbornou spolupráci i za čas, který mi věnoval při konzultacích.

Abstrakt

Ve své diplomové práci se zabývám vytvořením informačního systému správy vozového parku firmy BKB Metal a.s. Tato práce seznamuje s postupem vývoje celého informačního systému a ukazuje, co vše je k jeho vytvoření potřeba. Součástí práce je aplikace, která byla vytvořena pomocí aplikačního rámce Nette a také porovnání tohoto aplikačního rámce s jinými nejpoužívanějšími aplikačními rámci.

Klíčová slova: informační systém, správa vozidel, PHP, aplikační rámec Nette

Abstract

This dissertation deals with creating the information system of the car pool administration in the company BKB Metal a.s. It shows methods of creating the whole information system and demonstrates all the steps required for creating it. A part of this dissertation is the application which was created by using the Nette Framework as well as the comparison of this framework with other popular frameworks.

Key words: information system, car pool administration, PHP, Nette Framework

Seznam použitých zkratek

PHP	Hypertext Preprocessor (starší zkratka: Personal Home Page)
ERD	Entity-relationship diagram
PK	Primary key
FK	Foreign key
STD	State Transition Diagram
HTML	HyperText Markup Language
XHTML	Extensible HyperText Markup Language
XSS	Cross-site scripting
XSRF	Cross-Site RequestForgery
API	Application Programming Interface
OOP	Objektově orientované programování
MVC	Model-View-Controler
MVP	Model-View-Presenter
URL	Uniform ressource locator
URI	Uniform ressource identifier
NF	Nette Framework
QS	Quick Start
ZF	Zend Framework
CI	CodeInteger
AJAX	Asynchronous JavaScript and XML
DRY	Dont Repeat Yourself
KISS	Keep It Simple, Stupid!
ORM	Object-relational mapping
SEO	Search Engine Optimization
IDE	Integrated Development Environment

Seznam obrázků

Obsah

1. Úvod

Úkolem této diplomové práce je navrhnout a vytvořit aplikaci na základě požadavků firmy BKB Metal a.s. Jedná se o aplikaci určenou pro správu vozového parku této firmy. V této práci je postupně popsán vývoj celé aplikace, od prvotní konzultace až po zhodnocení výsledné aplikace.

Požadavky na tuto aplikaci byly dále doplněny o poznatky, které jsem nabral při studiu na Vysoké škole Báňské. Výsledná aplikace bude uložena na firemním serveru, tudíž k ní uživatelé budou moct přistupovat pouze v prostorách firmy. Aplikace bude vytvořena pomocí aplikačního rámce Nette. Abychom se více s tímto aplikačním rámcem seznámili, je součástí této práce také srovnání tohoto aplikačního rámce s dalšími nejpoužívanějšími.

1.1 Cíl práce

Cílem práce je vytvořit aplikaci, díky které si bude moct firma BKB Metal a.s. vést evidence o svých vozidlech, jednotlivých jízdách a dalších datech, které potřebuje. Aplikace bude víceuživatelská, jelikož musí být rozlišeny role a s nimi spjatá práva pro přístup k jednotlivým funkcím aplikace. S celou aplikací a všemi funkcemi bude moct pracovat uživatel s rolí „Admin“. Omezené využití aplikace bude mít uživatel s rolí „User“ a poslední rolí bude „Nepřihlášený uživatel“, kterému bude zobrazen pouze přihlašovací formulář.

V rámci řešení této diplomové práce je také analýza a zhodnocení použitého PHP aplikačního rámce Nette, pomocí kterého je aplikace správy vozového parku pro firmu BKB Metal a.s. implementována. Aplikační rámce zvyšují efektivitu práce programátora, řeší bezpečnost aplikace a díky oddělení aplikační logiky od návrhu vizuální podoby se aplikace stávají snáze udržitelnými. Součástí této práce je také srovnání PHP aplikačního rámce Nette s jinými nejpoužívanějšími PHP aplikačními rámci.

2. Analýza potřeb firmy BKB Metal a.s.

Společnost BKB Metal a.s. využívá vlastní informační systém, který byl vytvořen přímo na míru podle potřeb společnosti. Současný informační systém pokrývá oblast archivace technické dokumentace, účetnictví, manažerského účetnictví, zakázkového systému, docházky a pošty. V době zadání diplomové práce nebyla součástí informačního systému správa vozového parku firmy. Vzhledem k tomu, že společnost BKB Metal a.s. nutně takovýto systém potřebuje, vznikl požadavek na aplikaci pro správu vozového parku šitou na míru této společnosti. Společnost potřebuje mít také plná práva k výsledné aplikaci a podporu správy aplikace autorem s možností rozšíření do budoucna.

Prvním krokem analýzy potřeb bylo zjištění přesných požadavků firmy BKB Metal a.s. Na úvodní schůzce byl domluven obsah celé aplikace a způsob zpracování.

2.1 Role v aplikaci

Po konzultaci o rozdělení zaměstnanců ve firmě byly domluveny dvě základní role. Všechna práva bude mít role „Admin“, kterou budou zastávat vedoucí pozice ve firmě, sekretářka, účetní atd. Záleží pouze na vedení firmy, komu se rozhodnou přidělit roli „Admin“. Druhou rolí bude „User“, tedy klasický uživatel systému, který má oproti „Adminovi“ značně omezené možnosti. Všechna oprávnění těchto rolí jsou popsána v Programátorské příručce (Příloha A).

Nesmí být zapomenuto na jednu velice důležitou roli, kterou je „Nepřihlášený uživatel“. Tomu bude zobrazen pouze přihlašovací formulář. Aby se „Nepřihlášený uživatel“ mohl do aplikace přihlásit, musí mít svůj vlastní účet, který může vytvořit pouze uživatel s rolí „Admin“.

2.2 Obsah aplikace

Jak již samotný název diplomové práce napovídá, jedná se o správu vozového parku. Muselo být přesně stanoveno, které informace chce firma v systému uchovávat. Základním prvkem ve firmě jsou divize. Jednotlivým divizím jsou přiřazeni zaměstnanci a vozidla. Zaměstnancům budou přiřazeny role podle jejich postavení ve firmě. Dále musí aplikace obsahovat knihu jízd a rezervace pro jednotlivá vozidla. Velice důležitá data pro firmu jsou vlastnosti vozidla: STK a pojištění. Aplikace by měla s dostatečným předstihem hlásit, zda v blízké době dojde k vypršení platnosti u těchto vlastností. Dále u vozidla mohou nastat komplikace v podobě závady. Každá závada musí být evidovaná, a pokud není v zaměstnancových silách ji opravit, musí být vozidlo odvezeno do servisu. Systém musí hlídat vztahy mezi vozidlem v servise vzhledem k rezervaci vozidel, aby nemohlo dojít k zapsání rezervace v době, kdy vozidlo není možné použít. Poslední důležitou informací budou prohlídky vozidel. Nyní jsou stanovena všechna potřebná data pro vznik aplikace. Dalším krokem jsou podrobnější informace o jednotlivých prvcích. Zpracování takovýchto informací je popsáno v kapitole 3.

2.3 Umístění aplikace

S vedením firmy bylo dohodnuto, že aplikace bude umístěna na serveru firmy a uživatel se do ní bude moct přihlásit pouze v prostorách firmy.

2.4 Způsob přihlášení do aplikace

Do aplikace se zaměstnanci přihlašují pomoci loginu a hesla, které náleží jejich uživatelskému účtu. Ten zaměstnancům předem vytvoří uživatel s rolí „Admin“. Jako login bude sloužit emailová adresa zaměstnanců firmy.

3. Specifikace zadání

Po ukončení analýzy potřeb firmy musí být specifikováno zadání vytvářené aplikace. Zadání je základní popis řešení budoucího díla. Musí být přesně, úplně a bezesporně specifikováno. Zadavatelem je v tomto případě firma BKB Metal a.s. Případné přeprogramování systému v rámci špatně definovaného zadání se může zadavateli velice prodražit. Pokud zadavatel neuvede všechny potřebné údaje, musí se na ně analytik (příjemce zadání) doptat, aby bylo kompletní. Zadání má dvě základní části [1]: Funkční požadavky a Nefunkční požadavky.

3.1 Funkční požadavky

Funkčními požadavky rozumíme požadavky na věcný a problémový obsah systému. Analytik má připravenou šablonu otázek, na které se zadavatele ptá, nebo mu je dá zapsat. Základní struktura takovéto šablony je: proč, k čemu, kdo, vstupy, výstupy a funkce.

3.1.1 Proč je zapotřebí nová aplikace

Tato zdánlivě jednoduchá otázka většinou uvede zadavatele do rozpaků. Úkolem je popsat okolnosti rozhodnutí o vytvoření nové aplikace. Je možné, že firma již fungující aplikaci má. V tom případě musí analytik zjistit, proč jim tato aplikace nevyhovuje.

Společnost BKB Metal a.s. má již plně fungující informační systém, ve kterém chybí možnost pro správu vozového parku. Proto vznikl požadavek na vytvoření nové aplikace pro tento účel.

3.1.2 K čemu bude aplikace sloužit

Tato otázka dá odpověď na to, co je hlavní prioritou celé aplikace: vnitřní evidence či vnější representace.

Aplikace pro firmu BKB Metal a.s. bude sloužit pro evidenci a správu vozového parku. Evidovat se budou informace o vozidlech, divizích, zaměstnancích, jízdách, závadách, rezervacích, servisech, pojištěních, STK a prohlídkách.

3.1.3 Kdo bude s aplikací pracovat

Odpověď má definovat, kteří uživatelé budou se systémem pracovat, zda se budou dělit do rolí a případně jaké budou mít role práva.

Aplikace pro správu vozového parku bude mít dvě základní role, které budou mít různá práva pro práci v systému. Nesmí se zapomenout na roli „Nepřihlášený uživatel“, která bude mít možnost zobrazit pouze přihlašovací obrazovku. Tyto role již byly definovány v kapitole 2.1.

3.1.4 Jaké budou vstupy do systému

Vstupy jsou informace, neboli seznam entit, které mají být v aplikaci evidované. Vstupy by měli být slovně popsány. Jejich podrobným rozborem se zabývá kapitola 3.

Vstupy pro aplikaci společnosti BKB Metal a.s.:

1) Budou se vyplňovat informace o jednotlivých zaměstnancích
2) Pro divize budeme potřebovat vědět pouze její název, zkratku a stav
3) Vozidla budou hlavním informačním prvkem a většina tabulek bude směřována na doplňující informace o vozidlech
4) Kniha jízd bude obsahovat záznamy o provedených jízdách
5) V systému bude evidence rezervací vozidel
6) U závad budeme potřebovat informace o tom, na kterém vozidle závada nastala a informace o ni
7) Pro servis budeme potřebovat data o vozidlu a informace o samotném servise na něm provedeném. Může také vzniknout v návaznosti na závadu.
8) Pro popis pojištění budeme potřebovat evidenci o tom, jaké pojištění vozidlo má a dobu jeho platnosti
9) U STK budeme potřebovat informace o provedení a hlavně o době platnosti
10) V prohlídkách budeme evidovat typ prohlídky a informace o ni

3.1.5 Jaké budou výstupy ze systému

Výstupy znamenají všechny výstupní sestavy, které mohou uživatelé potřebovat. Pro aplikaci správy vozového parku jsou to tyto výstupy:

1) Vlastnosti jednotlivých vozidel
2) Seznam zaměstnanců, které společnost má
3) Výpis divizí
4) Podrobná kniha jízd
5) Výpis rezervací
6) Seznam závad
7) Výpis pojištění
8) Výpis STK
9) Seznam servisů na vozidlech
10) Jednotlivé prohlídky na vozidlech

3.1.6 Jaké funkce bude aplikace plnit

Tyto informace slouží pro výpis všech funkcí, které se budou moct v aplikaci provádět. Jejich rozborem se zabývá Funkční analýza, která je zpracována v kapitole 4.2. Zde si uvedeme 7 příkladů funkcí z aplikace pro správu vozového parku:

1) Přidání divize
2) Editace divize
3) Výpis divizí
4) Přidání zaměstnance
5) Editace zaměstnance
6) Výpis daného zaměstnance
7) Výpis všech zaměstnanců

3.2 Nefunkční požadavky

Kromě věcné náplně aplikace musí být také uvedeny další velice důležité informace. Tyto informace se dělí do kategorií pro specifikaci priorit, způsoby řešení a nefunkční požadavky. Tyto požadavky se nevyužívají při analýze aplikace. Jsou ale velice důležitým prvkem pro uzavírání smluv.

3.2.1 Požadavky na priority výsledné aplikace

Mezi priority výsledné aplikace zahrnujeme efektivitu, rychlost, přehlednost, pohodlnost, bezpečnost, spolehlivost a složitost aplikace. Jedná se o velice důležité prvky, které jsou jedním z hlavních měřítek pro stanovení ceny aplikace.

Firma BKB Metal a.s. má hlavní požadavky na spolehlivost a přehlednost, pohodlnost a bezpečnost jejich aplikace.

3.2.2 Požadavky na způsob řešení

Požadavky na způsob řešení definují prostředí a programovací jazyk pro vytvoření aplikace, možnosti využití standardů a jak bude zpracována dokumentace. Dokumentace může být dvojího typu: uživatelská příručka a programátorská příručka.

Aplikace pro správu vozidel bude vytvořena pomocí PHP aplikačního rámce Nette. K aplikaci budou vytvořeny dvě základní příručky, tedy jak uživatelská příručka, tak programátorská příručka.

3.2.3 Vnější požadavky

Vnější požadavky specifikují cenu, časovou návaznost na tvorbu aplikace, legislativní omezení, firemní omezení a omezení v důsledku bezpečnosti. Také je zde stanoveno, zda bude aplikace navazovat
na již existující systém, nebo zda bude vystupovat samostatně.

Umístění aplikace firmy BKB Metal a.s. je již definováno v kapitole 2.3. Jedná se tedy o samostatný systém, který bude zpřístupněn pouze v prostorách firmy.

4. Návrh a analýza aplikace

Pokud je specifikace zadání kompletní, zahajuje se část návrhu a analýzy aplikace. Doplňují se detaily funkcí, datové struktury a algoritmy operací v aplikaci. Návrh a analýza aplikace se dělí na tři základní části: Datová analýza, Funkční analýza a Časová analýza.

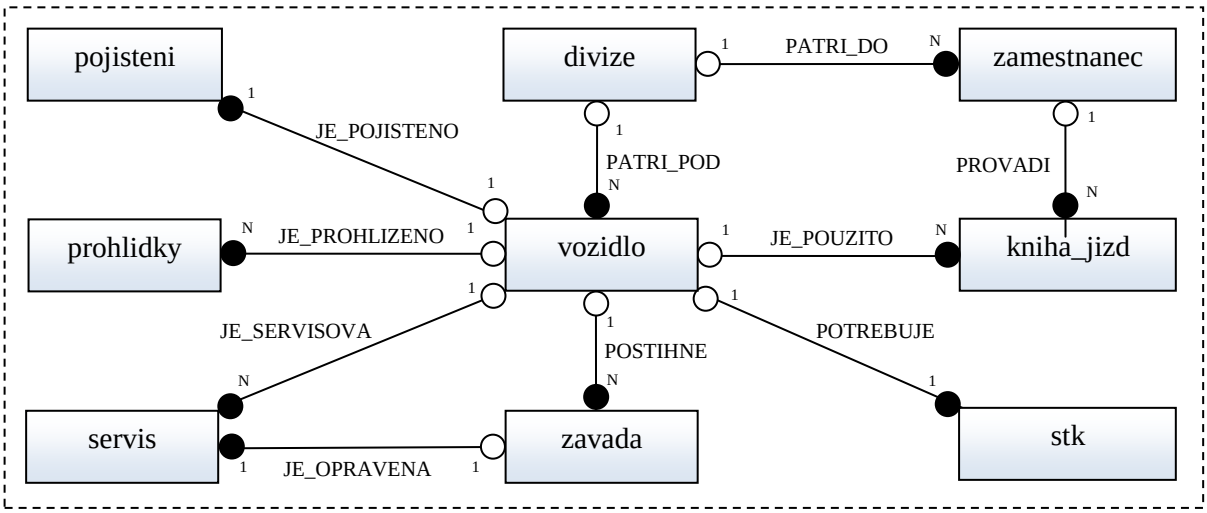
4.1 Datová analýza

Datová analýza je základem pro takové aplikace, kde se používá databáze a s ní spojené ukládání, vyhledávání, mazání a třízení dat. Každá takováto aplikace vychází ze správného návrhu struktury dat. Tato data musí být přesně popsána, specifikovány jejich vlastnosti a vztahy mezi nimi. Nejčastějšími prostředky [1] pro popis Datové analýzy je postupné použití těchto prostředků: ER diagramy, Lineální zápis typů entit, Popisy vztahů a Datový slovník.

4.1.1 ER diagram

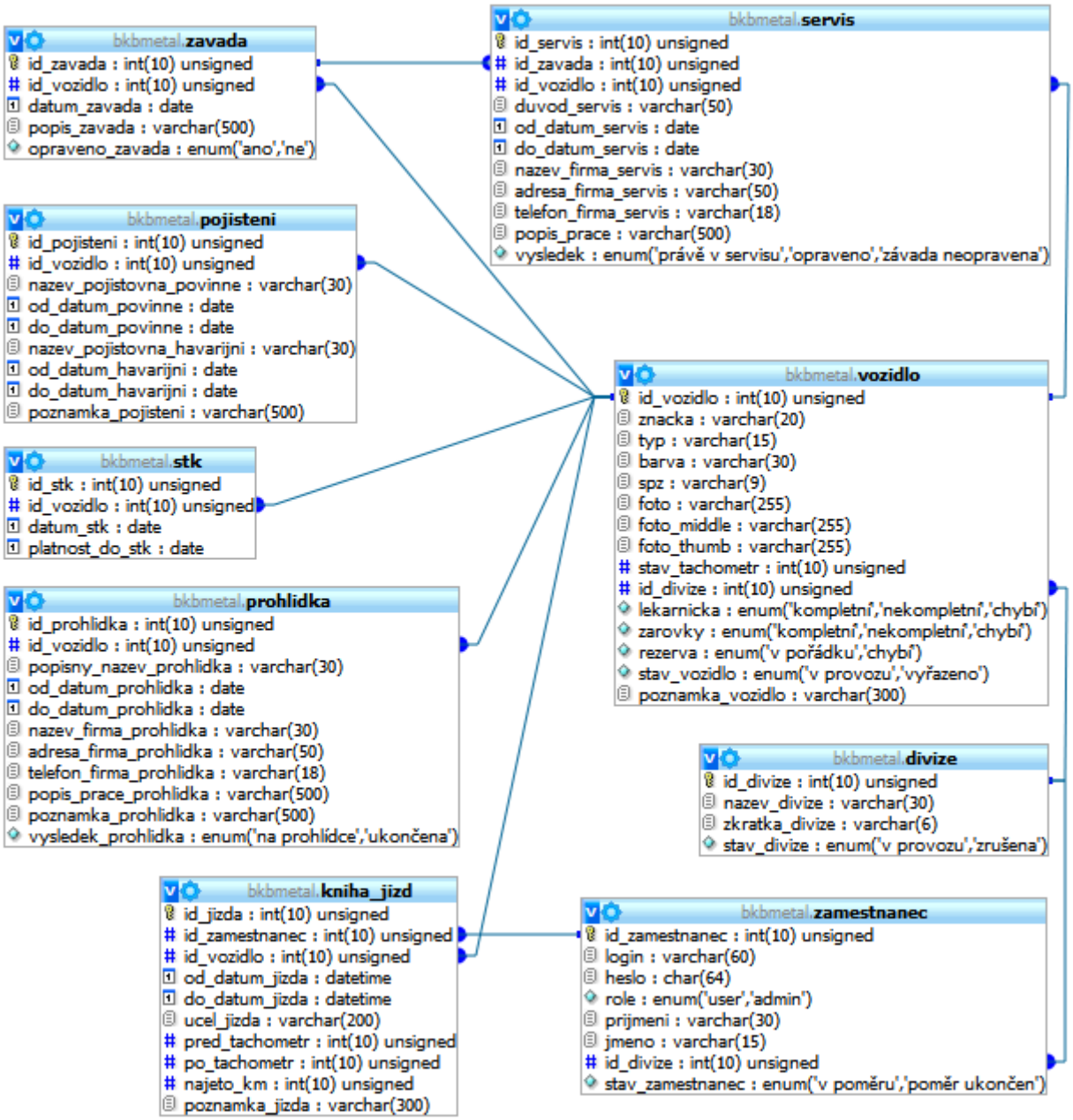
ER diagramy (ERD) jsou využívány pro abstraktní a konceptuální znázornění dat při analýze požadavků k popisu informační potřeby nebo typu informace uložené v databázi. Tyto diagramy se skládají z entit, tedy z objektů reálného světa, které jsou obsaženy v námi zpracovávané aplikaci. Tyto entity mají mezi sebou vztahy, které definují jejích návaznost. Vztahy mohou být typu: 1:1, 1:N nebo M:N. Dalším důležitým prvkem v těchto diagramech je znázornění kardinality. Ta nám určuje, které entity jsou vázány pevným poutem na jiné entity. Znázorňuje se za pomocí plných nebo prázdných bodů u propojení entit. Plný bod znázorňuje, že položka v této entitě může vzniknout pouze za předpokladu, že existuje položka v entitě vázané.

Aplikace pro firmu BKB Metal a.s. (Obr. 1) se skládá z devíti entit.



Obr 1: ER diagram databáze

Podrobnější diagram s výpisem jednotlivých atributů je graficky lépe zpracován, avšak základní ER diagram je přehlednější (Obr. 2).



Obr 2: ER diagram databáze s výpisem atributů

4.1.2 Lineární zápis typů entit

Lineární zápis typů entit slouží k přesnému popisu atributů pro jednotlivé entity. Nejdůležitějším popisem v tomto zápisu je stanovení primárních a cizích klíčů. Právě primární a cizí klíče nám slouží k propojení jednotlivých entit. Primárním klíčem nejčastěji bývá identifikace celé entity. Pro lepší

pochopení jsou primární klíče vyznačeny tučným písmem a cizí klíče kurzívou. Délka jednotlivých atributů a jejich datové typy jsou popsány v kapitole 3.1.4.

[primary key, *foreign key*]

1) divize (**id_divize** , nazev_divize, zkratka_divize, stav_divize)
2) zamestnanec (**id_zamestnanec** , login, heslo, role, prameni, jmeno, *id_divize*, stav_zamestnanec)
3) vozidlo (**id_vozidlo** , znacka, typ, barva, spz, foto, foto_middle, foto_thumb, stav_tachometr, *id_divize*, poznamka_vozidlo, lekarnicka, zarovky, rezerva, stav_vozidlo)
4) kniha_jizd (**id_jizda**, *id_zamestnanec*, *id_vozidlo*, od_datum_jizda, do_datum_jizda, udel_jizda, pred_tachometr, po_tachometr, najeto_km, poznamka_jizda)
5) zavada (**id_zavada** , *id_vozidlo*, datum_zavada, popis_zavada, opraveno_zavada)
6) servis (**id_servis** , *id_zavada*, *id_vozidlo*, duvod_servis, od_datum_servis, do_datum_servis, nazev_firma_servis, adresa_firma_servis, telefon_firma_servis, popis_prace, vysledek)
7) pojisteni (**id_pojisteni** , *id_vozidlo*, pojišťovna_povinne, od_datum_povinne, do_datum_povinne, pojišťovna_havarijni, od_datum_havarijni, do_datum_havarijni, poznamka_pojisteni)
8) stk (**id_stk** , *id_vozidlo* , datum_stk, platnost_do_stk)
9) prohlidka (**id_prohlidka** , *id_vozidlo*, popisny_nazev_prohlidka, od_datum_prohlidka, do_datum_prohlidka, nazev_firma_prohlidka, adresa_firma_prohlidka, telefon_firma_prohlidka, popis_prace_prohlidka, poznamka_prohlidka, vysledek_prohlidka)

4.1.3 Popis vztahů

Velice důležitý je popis vztahů mezi jednotlivými entitami. Jak již bylo výše uvedeno, mohou být typu: 1:1, 1:N nebo M:N. Pro lepší pochopení vztahů mezi entitami si uvedeme příklady:

- Vztah 1:1 – jedno vozidlo má pouze jedno STK
- Vztah 1:N – jedno vozidlo je zapsáno v knize jízd několikrát pro různé jízdy
- Vztah M:N – tento vztah není v aplikaci pro správu vozového parku využit, ale příklad si uvedeme: je zadáno několik *firem*, a ty vyrábějí stejné *výrobky*. Vztah *firma* vyrábí *výrobek* je vztah M:N.

Nyní vypíšeme vztahy mezi tabulkami v aplikaci pro správu vozového parku:

JE_PROHLIZENO	(prohlidky, vozidlo)	N : 1
PATRI_DO	(divize, zamestnanec)	1 : N
PATRI_POD	(divize, vozidlo)	1 : N
JE_OPRAVENA	(servis, zavada)	1 : 1
POSTIHNE	(vozidlo, zavada)	1 : N
JE_POJISTENO	(pojisteni, vozidlo)	1 : 1

JE_POUZITO	(vozidlo, kniha_jizd)	1 : N
PROVADI	(zamestnanec, kniha_jizd)	1 : N
POTREBUJE	(vozidlo, stk)	1 : 1
JE_SERVISOVANO	(vozidlo, servis)	1 : N

4.1.4 Datový slovník

Datový slovník přesně specifikuje, v jakém datovém typu budou která data uložena. S tím souvisí jejich maximální délka, a zda můžou být nulová, či nikoliv. Pokud se jedná o primární klíč, určitě musí mít svou vlastní indexaci a jeho značení je PK (Primary key). Cizí klíč musí být také přesně označen zkratkou FK (Foreign key). Nakonec se uvádí popis toho, co vlastně jednotlivé atributy dané entity ve slovníku znamenají. V této části nejsou vypsány všechny slovníky, ale jsou uvedeny pouze tři příklady. Zbylé datové slovníky jsou umístěny v programátorské příručce (Příloha A).

zamestnanec

Atribut	Datový typ	Klíč	Null	Index	Popis
id_zamestnanec	int(10)	PK	ne	ano	Identifikace zaměstnance
login	varchar(60)		ne		Login zaměstnance
heslo	char(64)		ne		Heslo zaměstnance
role	enum('user', 'admin')		ne		Role zaměstnance
prijmeni	varchar(30)		ne		Příjmení zaměstnance
jmeno	varchar(15)		ne		Jméno zaměstnance
id_divize	int(10)	FK	ne		Identifikace divize
stav_zamestnanec	enum('v poměru', 'poměr ukončen')		ne		Stav zaměstnance

pojisteni

Atribut	Datový typ	Klíč	Null	Index	Popis
id_pojisteni	int(10)	PK	ne	ano	Identifikace pojisteni
id_vozidlo	int(10)	FK	ne		Identifikace vozidla
pojistovna_povninne	varchar(30)		ne		Pojišťovna pro povinné pojištění
od_datum_povinne	date		ne		Datum od povinného poj.
do_datum_povinne	date		ne		Datum do povinného poj.
nazev_pojistovna_havarijni	varchar(30)		ano		Pojišťovna havarijního poj.
od_datum_havarijni	date		ano		Datum od havarijního poj.
do_datum_havarijni	date		ano		Datum do havarijního poj.
poznamka_pojisteni	varchar(500)		ano		Poznámka pojištění

divize

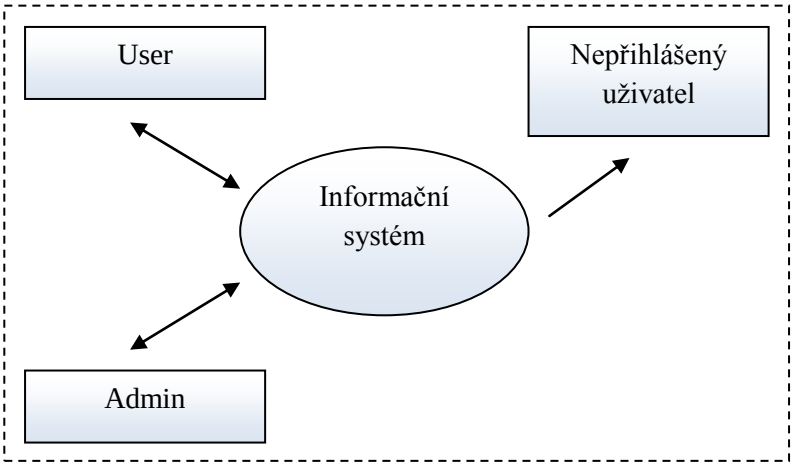
Atribut	Datový typ	Klíč	Null	Index	Popis
id_divize	int(10)	PK	ne	ano	Identifikace divize
nazev_divize	varchar(30)		ne		Název divize
zkratka_divize	varchar(6)		ne		Zkratka divize
stav_divize	enum('v provozu','zrušena')		ne		Stav divize

4.2 Funkční analýza

Funkční analýza má za úkol popsat všechny operace, které se budou s daty provádět. Obecně jde o operace ukládání, modifikace, mazání, výpočty, třízení a vyhledávání dat. Každá operace musí být podrobně zpracována tak, aby bylo předem stanoveno, co se ve kterém kroku operace provede. Mezi nejčastěji využívané prvky [1] této analýzy patří: Kontextové diagramy, Datové toky a Minispecifikace.

4.2.1 Kontextové diagramy

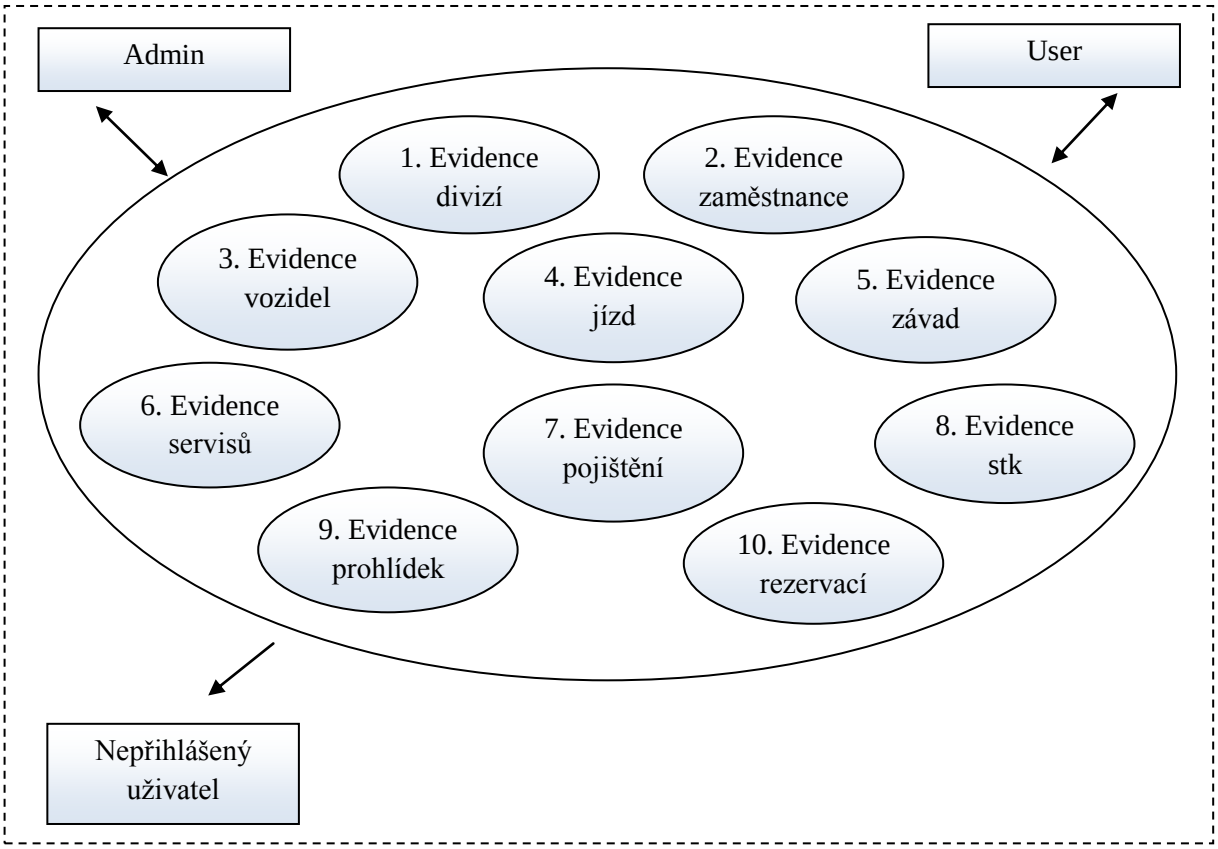
Pro pochopení funkčnosti aplikace jsou využívány kontextové diagramy. Celou aplikaci nelze dopodrobna zobrazit za pomocí jednoho diagramu. Proto se tyto diagramy dělí do úrovní. Nejvyšší úrovní je základní kontextový diagram (Obr. 3). Ten obsahuje celý systém nebo aplikaci jako jednu funkci. Jsou zde definovány hranice systému a všichni aktéři, kteří budou v tomto systému či aplikaci vystupovat.



Obr 3: Základní úroveň kontextového diagramu

Další úroveň je vytvořena bezprostředním rozkladem (neboli dekompozicí) základní úrovně kontextového diagramu. Z takto vytvořené úrovně lze již vyčíst, ke kterým částem aplikace má která role přístup. Jedná se o 0.úroveň. Nikdy nesmí být opomenuto na „Nepřihlášeného uživatele“. V aplikaci firmy BKB Metal a.s. nebude mít žádnou možnou proveditelnou funkci. Obecně ale platí, že i takovýto uživatel může do systému zasahovat. Uvedeme si příklad: vytvořili jsme internetové stránky, které budou mít dvě role pro přihlášení. Na tyto stránky umístíme anketu, do které bude moct uživatel hlasovat, aniž by se přihlásil. V tomto případě může hlasovat právě „Nepřihlášený uživatel“ a musíme tedy počítat s jeho možnou účastí na hlasování.

V této aplikaci nebude moct „Nepřihlášený uživatel“ vykonávat žádnou operaci, jak je patrné ze základního kontextového diagramu. Tuto situaci popisuje šipka, která směřuje pouze ven. Poznáme to také z diagramu 0.úrovně (Obr. 4), kdy „Nepřihlášenému uživateli“ není přiřazen žádný možný podíl na operacích pro jednotlivé části aplikace.



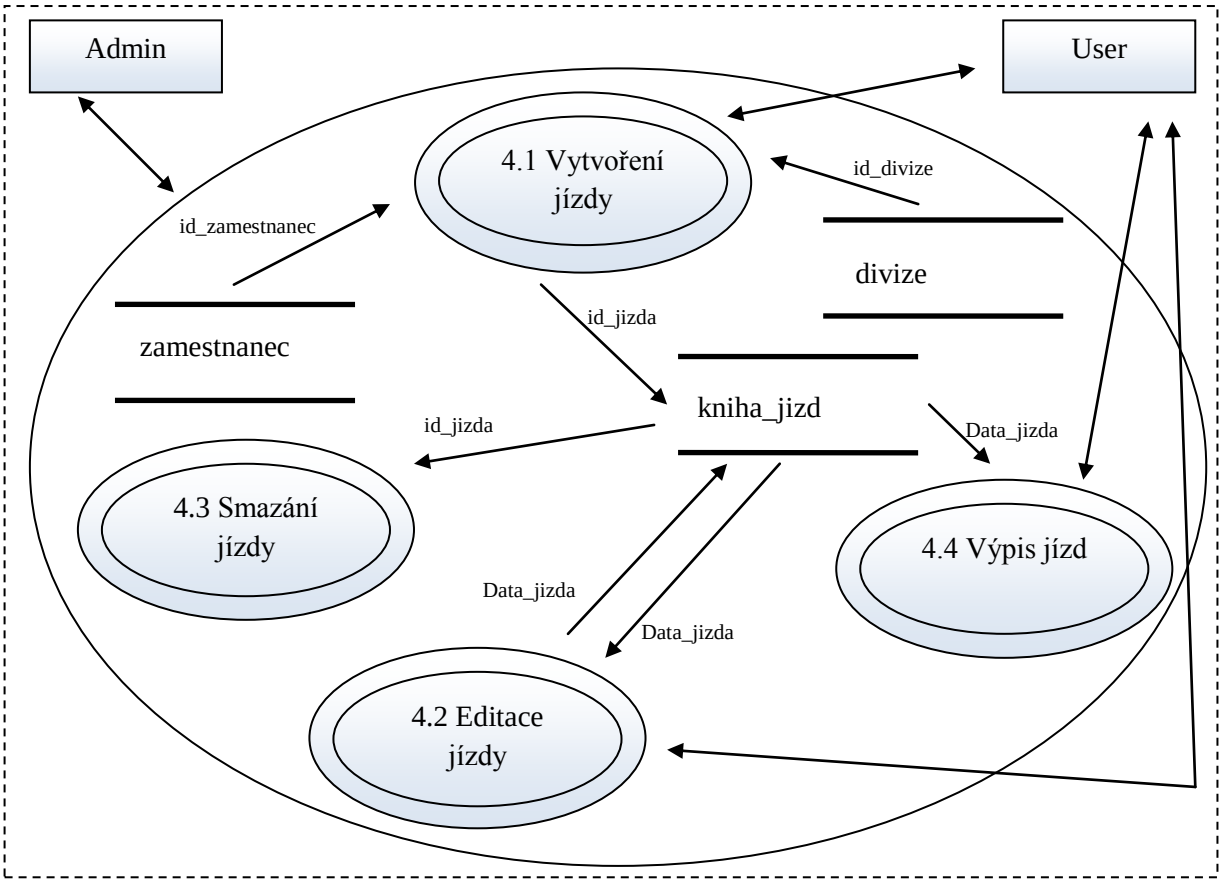
Obr 4: 0.úroveň kontextového diagramu

V poslední použité úrovni, tedy v kontextovém diagramu 1.úrovně, je již přímo rozepsáno, co je ve které z evidencí prováděno. Je zde znázorněn aktér, který může dané operace provádět. V případě, že je některá z funkcí prováděna automaticky aplikací, tak jako aktér vystupuje *SYSTEM*.

Důležité je, aby šlo názorně vidět, se kterými daty je ve které operaci pracováno a ze které tabulky jsou data načítána. Pokud je do databáze prvek přidáván, je tato operace znázorněna přidáním nového indexu. Pokud je prvek v databázi upravován, musí být data z databáze nejprve načtena a poté mohou být editovaná data vložena zpět na stejný řádek se stejným indexem, který nebude nikdy editován. Pokud jsou data z databáze mazána, je tento proces znázorněn odstraněním indexu. A nakonec, pokud jsou data z databáze pro danou tabulku vypisována, je zvolen výpis za pomocí datového toku. Datové toky a jejich vlastnosti jsou popsány v kapitole 3.2.2.

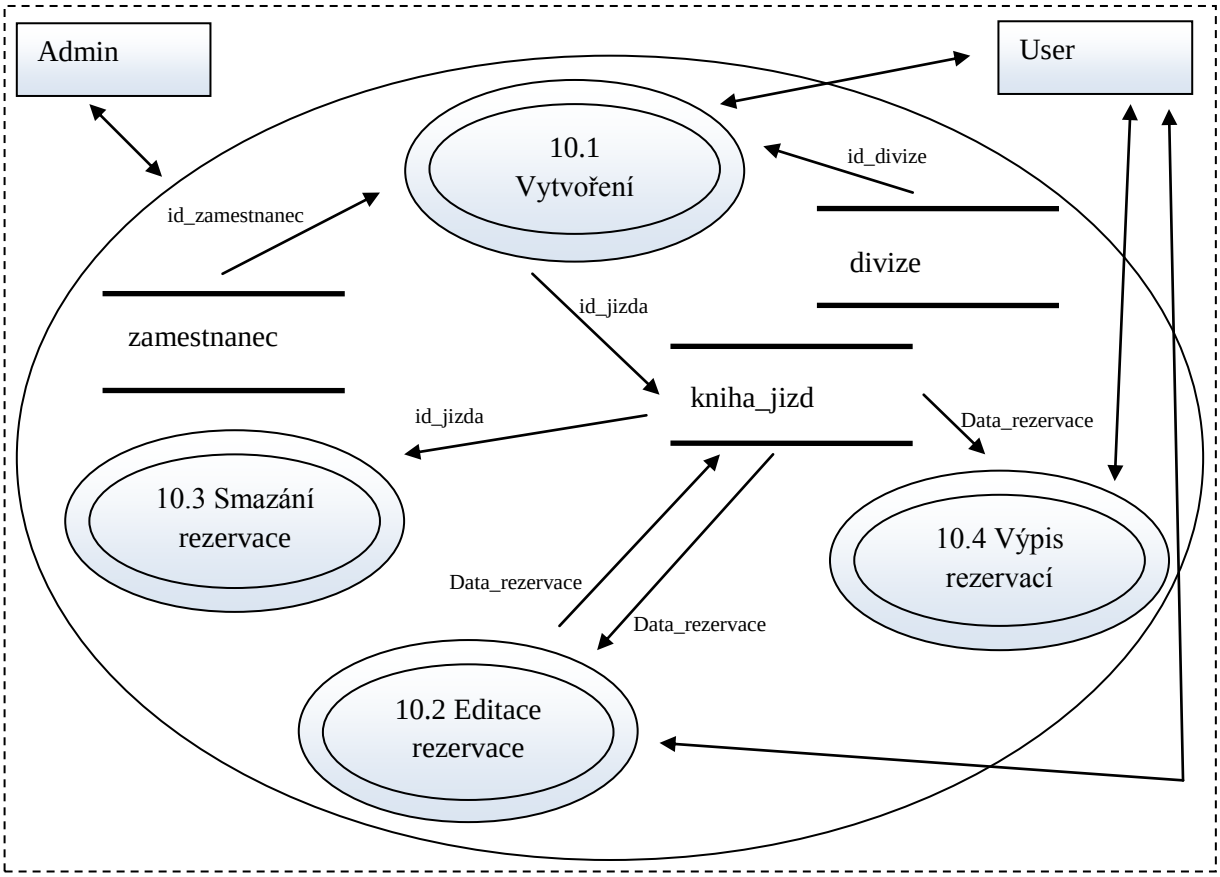
Zde je příklad 2 diagramů (Obr. 5 a 6). Všechny diagramy pro tuto aplikaci jsou zpracovány v jiné části práce (Příloha A).

Evidence jízd



Obr 5: 1.úroveň kontextového diagramu pro evidenci jízd

Evidence rezervací



Obr 6: 1.úroveň kontextového diagramu pro evidenci rezervací

4.2.2 Výpis datových toků

Datový tok vyjadřuje přesun dat nebo informací z jedné části systému do jiné, z okolí systému do systému nebo ze systému do jeho okolí. Je znázorněn šipkou, která znamená směr toku dat. Laicky řečeno nám šipky znázorňují, která data jsou z tabulky vybírána při které operaci. Vysvětlíme si to na příkladu: ve firmě máme tabulku *zamestnanec*. Pokud chceme vypsat všechny zaměstnance, vybereme všechny položky tabulky. Pokud ale chceme zobrazit, který uživatel je momentálně přihlášen, potřebujeme z tabulky *zamestnanec* pouze jeho *id_zamestnanec, prijmeni, jmeno a roli*. Nyní si znázorníme tyto dva datové toky:

Data_zam1

Atribut	Datový typ	Klíč	Null	Index	Popis
id_zamestnanec	int(11)	PK	ne	ano	Identifikace zaměstnance
prijmeni	varchar(30)		ne		Příjmení zaměstnance
jmeno	varchar(15)		ne		Jméno zaměstnance
role	enum(‘user’,’admin’)		ne		Role zaměstnance

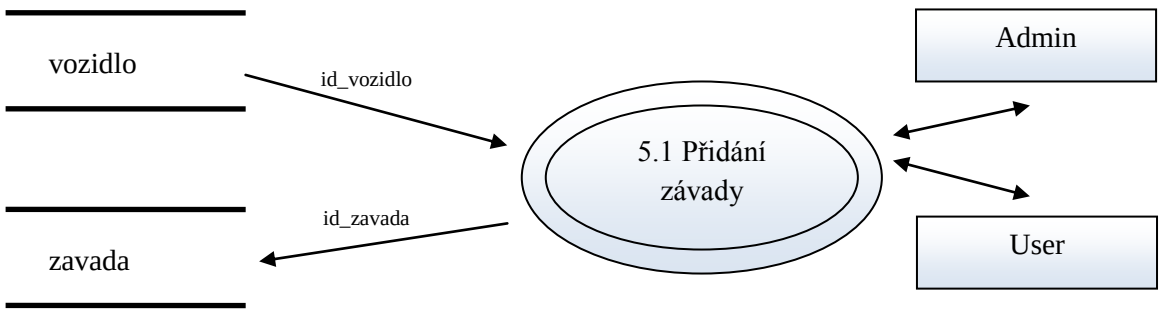
Data_zam2

Atribut	Datový typ	Klíč	Null	Index	Popis
id_zamestnanec	int(10)	PK	ne	ano	Identifikace zaměstnance
login	varchar(60)		ne		Login zaměstnance
heslo	char(64)		ne		Heslo zaměstnance
role	enum('user', 'admin')		ne		Role zaměstnace
prijmeni	varchar(30)		ne		Příjmení zaměstnance
jmeno	varchar(15)		ne		Jméno zaměstnance
id_divize	int(10)	FK	ne		Identifikace divize
stav_zamestnanec	enum('v poměru', 'poměr ukončen')		ne		Stav zaměstnance

4.2.3 Minispecifikace

Minispecifikace je popis elementární funkce – tedy funkce na nejnižší úrovni. Slouží k úplnému popisu jednotlivých operací v celém systému. Obsahuje část kontextového diagramu, ve kterém jsou znázorněny datové toky a aktéři. Dále obsahuje náhled na formulář pro provedení operace, ve které jsou přesně zapsány prvky z tabulek. Minispecifikace také musí obsahovat popis algoritmu, který ukazuje, co vše tato operace musí vykonat krok po kroku. Pokud se jedná o složitou operaci, mohou být dále ještě dopsány doplňující poznámky, které dopodrobna specifikují celou operaci. Nyní si uvedeme příklady čtyř minispecifikací. Tyto příklady znázorňují čtyři nejčastější operace, a to: přidávání objektu, mazání objektu, editace objektu a výpis objektu.

4.2.3.1 Přidání závady

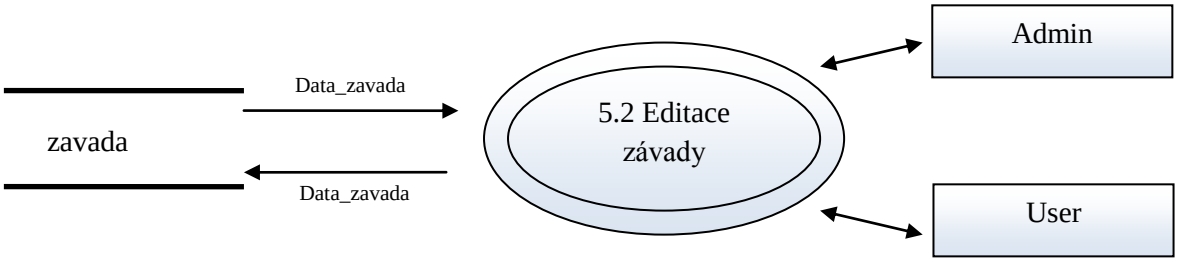


NOVÁ ZÁVADA:	
Číslo závady: id_zavada	Vozidlo: id_vozidlo
Datum: datum_zavada	Popis: popis_zavada
Opraveno: opraveno_zavada	

Algoritmus

1. Uživatel vybere volbu Přidat závadu
2. IS vygeneruje jednoznačné číslo závady do *id_zavada*
3. Z rolovacího menu uživatel vybere, pro které vozidlo závada vnikla, informace se uloží do *id_vozidlo*
4. Do *datum_zavada* vloží uživatel datum vzniku závady
5. Do *popis_zavada* vloží uživatel popis závady
6. Do *opraveno_zavada* vloží uživatel informaci o tom, zda byla závada opravena nebo ne

4.2.3.2 Editace závady



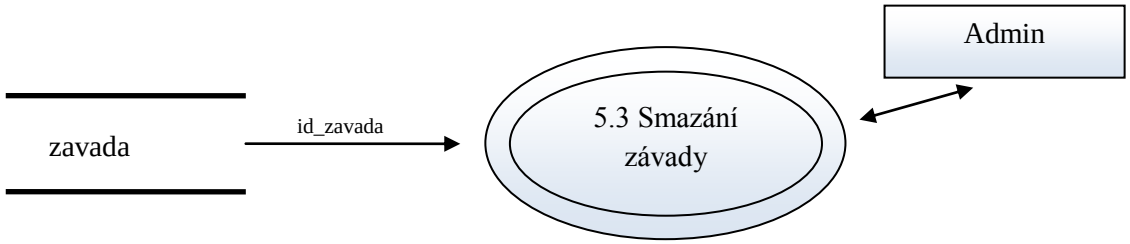
VÝPIS ZÁVAD:					
Číslo: Č.	Vozidlo: id_vozidlo	Datum: datum_zavada	Popis: popis_zavada	Opraveno: opraveno_zavada	E S

EDITACE ZÁVADY:	
Vozidlo: id_vozidlo	Datum: datum_zavada
Popis: popis_zavada	Opraveno: opraveno_zavada

Algoritmus

1. Uživatel zvolí volbu zobrazení výpisu závad
2. Stiskne možnost: Editovat pro danou závadu
3. Zobrazí se formulář pro editaci, kde se do příslušných políček načtou data z databáze pro danou závadu podle *id_zavada*
4. Uživatel nyní edituje jednotlivá data, která potřebuje změnit
5. Po ukončení editace potvrdí uživatel uložení dat
6. *Id_zavada* zůstává stále stejné
7. *Do id_vozidlo* se vloží editovaná data
8. Do *datum_zavada* se vloží editovaná data
9. Do *popis_zavada se* vloží editovaná data
10. Do *opraveno_zavada* se vloží editovaná data

4.2.3.3 Smazání závady

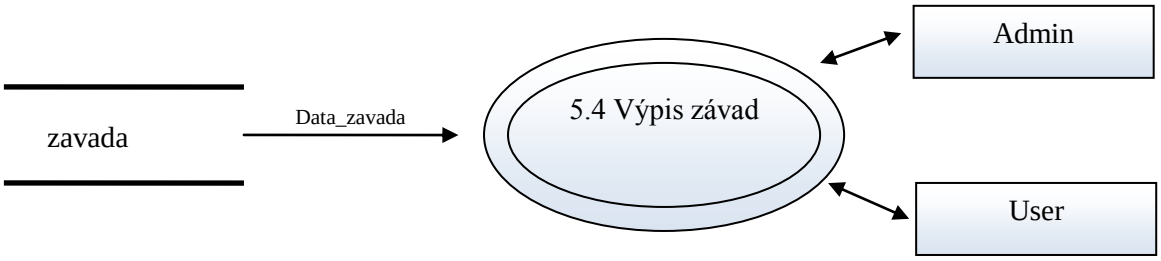


VÝPIS ZÁVAD:					
Číslo:	Vozidlo:	Datum:	Popis:	Opraveno:	
Č.	id_vozidlo	datum_zavada	popis_zavada	opraveno_zavada	E S

Algoritmus

1. Uživatel zvolí volbu zobrazení výpisu závad
2. Stiskne možnost: Smazat
3. Z databáze v tabulce závad bude smazána daná závada podle *id_zavada*

4.2.3.4 Výpis závad



VÝPIS ZÁVAD:					
Číslo:	Vozidlo:	Datum:	Popis:	Opraveno:	
Č.	id_vozidlo	datum_zavada	popis_zavada	opraveno_zavada	E S

Algoritmus

1. Uživatel zvolí volbu zobrazení výpisu všech závad
2. Načtou se všechny závady z databáze
3. Vypíšou se všechny závady z databáze do přehledné tabulky

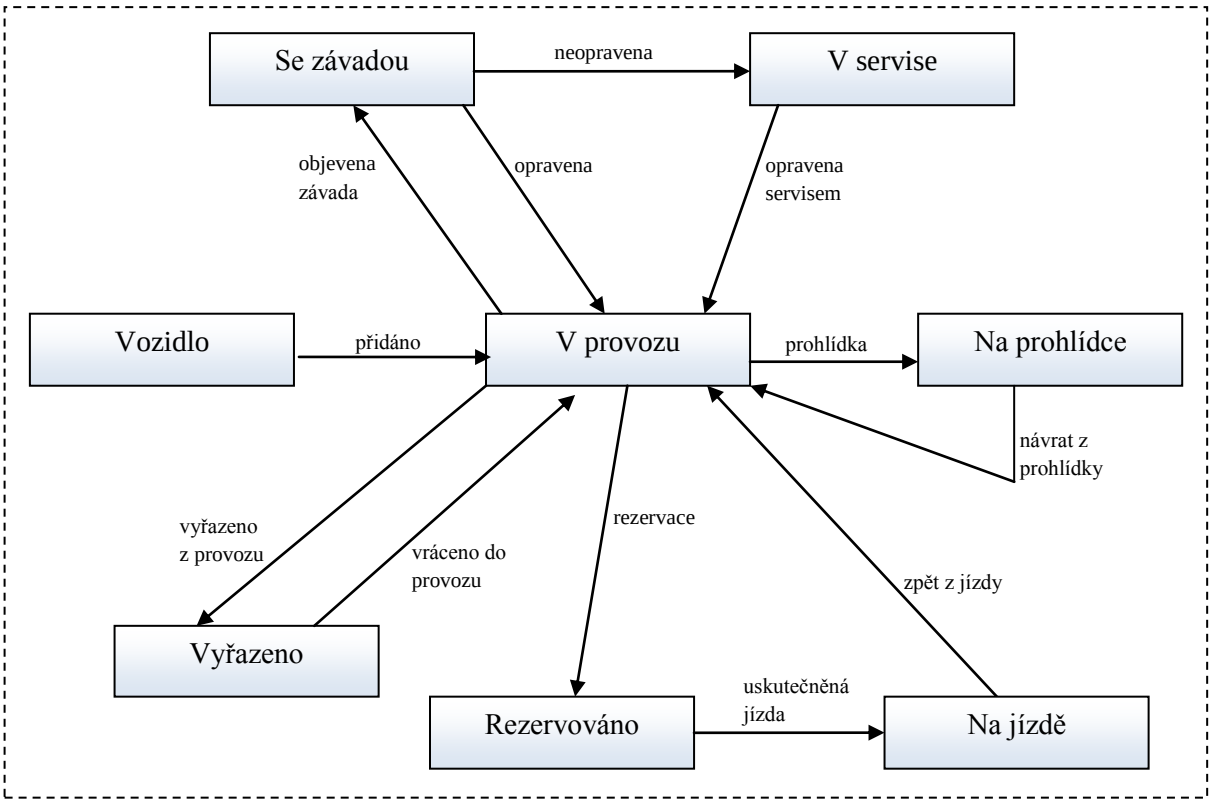
4.3 Časová analýza

Datová ani Funkční analýza neuvádí nic o časových návaznostech nebo posloupnostech jednotlivých prováděných funkcích. Obecně není možné spouštět jakoukoliv funkci kdykoliv (např. dokud není zapsáno vozidlo, není možné vytvořit jízdu). Pro přesnější stanovení časové návaznosti jsou využívány stavové diagramy [1].

4.3.1 Stavový diagram

Stavový diagram (STD) slouží k modelování systému v časových návaznostech, tedy v závislosti na čase nebo na pořadí funkcí. Modeluje chování systému v závislosti na působení vnějších událostí nebo na základě vnitřních změn stavů. Právě tyto stavy nám určují, kterou funkci systému může uživatel využít ve kterém stavu. Přechody mezi těmito stavy mohou být modifikace hodnot atributů, časová změna, vnitřní impuls systému nebo vnější impuls. STD je znázorňován pomocí grafu.

V aplikaci společnosti BKB Metal a.s. je nejdůležitějším prvkem vozidlo (Obr. 6), jehož stavy velice ovlivňují možnosti použití funkcí systému (např. pokud je vozidlo v servise, nemůže s ním být provedena jízda).



Obr 7: Stavový diagram pro vozidlo

5. Implementace aplikace

Implementace aplikace je praktický postup vytváření celé aplikace. Laicky lze říct, že vlastní implementace je převážně programátorskou prací. Postup implementace je přesně stanoven. Jednotlivé kroky nesmí být přeskakovány, jinak by došlo ke špatnému vytvoření celé aplikace, nebo by byla tvorba aplikace prodloužena.

Nejprve musí vytvořena databáze, do které mohou být následně vložena testovací data. Potom musí programátor aplikace stanovit přibližný vzhled aplikace a rozložení prvků v ní. Pokud se do aplikace budou uživatelé přihlašovat, musí být součástí aplikace přihlašovací formulář. Poté začne programátor vytvářet samotnou aplikaci. Do implementace spadají také všechny části, které byly v aplikaci zakomponovány, jako např. chybová hlášení nebo využití různých doplňujících prvků. Součástí implementace je také tvorba dokumentace, tedy uživatelské příručky a programátorské příručky.

5.1 Vytvoření databáze

Jedná se o prvotní krok celé implementace aplikace. Databáze je uspořádaná množina informací (dat) uložená na paměťovém médiu. Skládá se z tabulek obsahující atributy, které specifikují danou tabulku. Pro vytváření aplikace pro společnost BKB Metal a.s. je médiem firemní server. Databáze může být vytvořena za pomocí nástroje pro tvorbu a správu databází, např. phpMyAdmin [2], nebo je vytvářena jako posloupnost příkazů, které ji definují.

V této aplikaci byla databáze vytvořena jako kombinace obou možností. Nyní si některé tyto příkazy, které byly použity, popíšeme na příkladu z vytvářené aplikace pro divize:

```
CREATE TABLE IF NOT EXISTS `divize` (
  `id_divize` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `nazev_divize` varchar(30) CHARACTER SET utf8 NOT NULL,
  `zkratka_divize` varchar(6) CHARACTER SET utf8 NOT NULL,
  `stav_divize` enum('v provozu','zrušena') COLLATE utf8_bin NOT NULL,
  PRIMARY KEY (`id_divize`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin;
```

- **CREATE TABLE IF NOT EXISTS `divize` :** vytvoří tabulku s názvem „divize“
- **`id_divize` int(10) unsigned NOT NULL AUTO_INCREMENT :** vytvoří atribut „id_divize“ požadovaného datového typu s danou velikostí, určí, zda může být nulový či nikoli a nakonec mu nastaví automatické zvyšování hodnoty (číslovat začíná od 1).
- **`nazev_divize` varchar(30) CHARACTER SET utf8 NOT NULL :** postup jako v předešlé ukázce s tím rozdílem, že datový typ je pro text, tudíž musí být nastavena znaková sada, která určí, jaké vlastnosti text bude mít (utf8).

- **`stav_divize` enum(`v provozu`,`zrušena`) COLATE utf8_bin NOT NULL :** vytvoří atribut s výčtovým typem. Jako vstupy budou očekávány hodnoty uvedené v uvozovkách
- **PRIMARY KEY (`id_divize`) :** nastavení primárního klíče pro celou tabulku.
- **ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin :** doplňující vlastnosti pro celou tabulku, tedy typ uložiště a použití znakové sady.

5.2 Naplnění databáze daty

Databáze může být ihned po vytvoření naplněna daty. Jedná se buď o data, která jsou v aplikaci přesně stanovena a budou neměnná, nebo o data testovací. Taková data slouží jako pomocná data, díky kterým bude moct programátor aplikace ihned vidět výsledky výpisů tabulek a dalších operací.

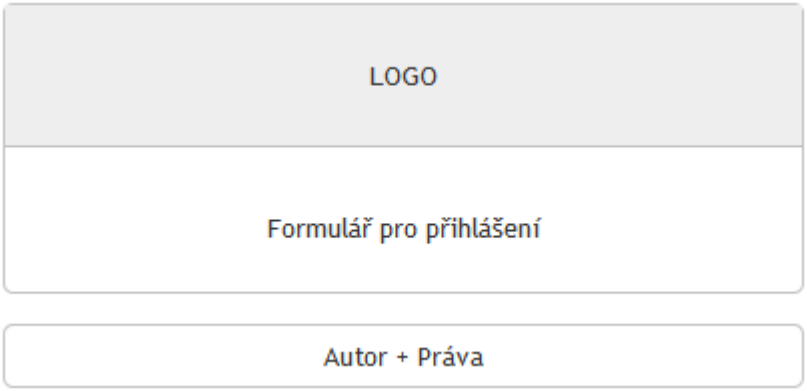
Databázi můžeme naplnit daty opět dvěma různými způsoby: použitím nástroje pro správu a tvorbu databází nebo pomocí posloupnosti příkazů. Jelikož jsou data pro potřeby testování mazána a editována, je doporučováno využití posloupnosti příkazů, které mohou testovací stav databáze kdykoliv obnovit. Zde je příklad naplnění tabulky databáze daty pomocí příkazů:

```
INSERT INTO `divize` (`id_divize`, `nazev_divize`, `zkratka_divize`, `stav_divize`) VALUES
(1, 'Vysoké pece', 'VP', 'v provozu'),
(2, 'Válcovny', 'VA', 'v provozu'),
(3, 'Průmyslové pece', 'PP', 'v provozu'),
(4, 'Energetika', 'EN', 'v provozu'),
(5, 'Ocelové konstrukce', 'OK', 'v provozu'),
(6, 'Režije', 'R', 'v provozu'),
(7, 'Představenstvo', 'P', 'v provozu');
```

- **INSERT INTO `divize` (`id_divize`, `nazev_divize`, `zkratka_divize`, `stav_divize`) VALUES :** určuje, do které tabulky a do kterých atributů se budou která data vkládat
- **(1, 'Vysoké pece', 'VP', 'v provozu') :** přesný popis dat pro jeden vložený řádek tabulky

5.3 Návrh vzhledu aplikace

Velice důležité je stanovení přibližného vzhledu aplikace ještě před jejím samotným vytvářením. Programátor pak může intuitivně umístit prvky, které bude vytvářet. Musí být vytvořen návrh pro všechny části aplikace, tedy jak pro část přihlašovací, tak pro část zobrazovací. Přihlašovací část musí obsahovat samotný formulář pro přihlášení, pokud se jedná o testovací verzi, tak informace o přihlašovacích údajích, a nakonec informace o autorovi a právech. Návrh pro přihlašovací část pro aplikaci společnosti BKB Metal a.s. (Obr. 8) je složen právě z takovýchto prvků.



Obr 8: Návrh vzhledu přihlašovací části

Návrh a rozmístění jednotlivých prvků aplikace by měl obsahovat:

- logo (případně další upřesňující informace)
- hlavní ovládací prvky či menu, které může být vytvářeno vodorovně nebo svisle
- zobrazovací část aplikace a případně její další rozvrstvení
- informace o právech a autorovi

Návrh pro aplikaci společnosti BKB Metal a.s. (Obr. 9) obsahuje podobné rozložení prvků.

Obr 9: Návrh a rozmístění prvků aplikace

5.4 *Vytvoření přihlašovacího formuláře*

Přihlašovací formulář obsahuje dvě položky pro vložení *loginu* a *hesla* uživatele. Vkládání hesla je vytvořeno tak, aby se nezobrazovalo vkládané heslo, ale pouze „hvězdičky“. Aby se mohl uživatel přihlásit, musí zadat přihlašovací údaje přesně tak, jak jsou uloženy v databázi. Při tvorbě přihlašovacího formuláře je velice důležitý postup kontroly zadaných údajů. Heslo totiž v databázi není uloženo jako text, ale jako *hash* [3]. Hash je posloupnost znaků, na které se původní text převede za pomocí *hashovaní funkce* [3]. Aby byl *hash* bezpečný, musí hashovaní funkce splňovat tyto vlastnosti:

- jakékoliv množství zadaných znaků vygeneruje stejně dlouhý *hash*
- malou změnou dat na vstupu způsobí velkou změnu na výstupu
- funkce je jednosměrná (nesmí být způsob jak se dostat k původní zprávě pokud známe *hash*)
- funkce je bez kolizí (nesmí dojít k tomu, že dvě rozdílné zprávy budou mít stejný *hash*)

Postup pro přihlášení do aplikace společnosti BKB Metal a.s. je následovný: uživatel zadá data do přihlašovacího formuláře a potvrdí zadaná data tlačítkem. Jako *login* slouží e-mail uživatele. Aplikace nejprve vyhledává v tabulce *zamestnanec,* zda v ní uživatel se zadaným *loginem* existuje. Pokud takového uživatele najde, vygeneruje ze zadaného hesla v přihlašovacím formuláři *hash,* a ten následně porovnává s *hashem*, které je uloženo v databázi pro již vyhledaného uživatele. Pokud se položky shodují, tak uživatel v systému existuje. Než ale aplikace uživateli přístup dovolí, musí ještě zkontrolovat, zda není uživatel ve stavu „poměr ukončen“. Pokud není, aplikace pustí uživatele dále.

5.5 *Zobrazení informací o přihlášeném uživateli*

Po přihlášení do aplikace musí být zobrazeny informace o tom, kdo je přihlášen a podrobnější výpis nejdůležitějších informací o přihlášeném uživateli. V zobrazení informací by měla být i možnost na přímou editaci přihlášeného uživatele, popřípadě alespoň editaci hesla. Dále zde musí být možnost pro odhlášení z aplikace.

V aplikaci pro společnost BKB Metal a.s. bude tato informace (Obr. 10) obsahovat jméno a příjmení přihlášeného uživatele, které budou přímým odkazem na editaci uživatele, divizi, roli a možnost pro odhlášení.

Obr 10: Informace o přihlášeném uživateli

5.6 *Umístění menu a loga*

Dalším postupem tvorby aplikace je umístění loga společnosti a menu. Jak již bylo v kapitole 5.3 uvedeno, tak logo a menu budou vedle sebe v horní části aplikace. Položky menu aplikace společnosti BKB Metal a.s. budou rozlišeny podle rolí přihlášeného uživatele. Uživatel přihlášený jako „User“ bude mít položky (Obr. 11): Kniha jízd, Vozidla, Rezervace, Závady a Zaměstnanci.

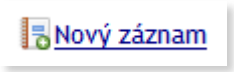
Obr 11: Vzhled menu a loga pro roli „User“

Uživatel s rolí „Admin“ bude moct provádět mnohem více operací, než uživatel s rolí „User“. Podle toho také vypadá počet položek v menu (Obr. 12): Kniha jízd, Vozidla, Rezervace, Pojištění, STK, Závady, Servis, Prohlídky, Zaměstnanci a Divize.

Obr 12: Vzhled menu pro roli „Admin“

5.7 *Vytvoření zobrazovací části*

Tato část aplikace je určena pro zobrazení všech výpisů a pro většinu operací v aplikaci. V levém horním rohu zobrazovací části je nadpis, který upřesňuje právě prováděnou operaci. V pravém horním rohu zobrazovací části je vyhrazeno místo pro zobrazení vybraných ovládacích prvků (Obr. 13), jako je například vkládání dat pro právě zobrazenou položku menu.



Obr 13: Ovládací prvek pro přidání záznamu do Knihy jízd

Pod výše zmíněným nadpisem a ovládacími prvky se nachází hlavní zobrazovací část, ve které se data vypisují, vkládají, upravují či mažou. Většina výpisů bude prováděna právě v této části aplikace (Obr. 14).

Obr 14: Příklad výpisu Knihy jízd pro roli „Admin“

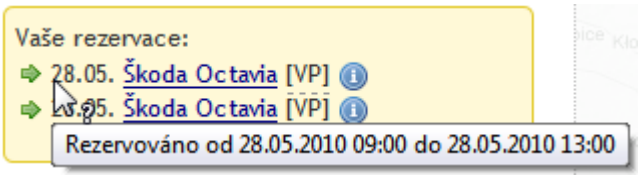
5.8 *Vytvoření informačního panelu*

Samotný informační panel je v této aplikaci pro společnost BKB Metal a.s. velice důležitý. Budou zde zobrazována různá varování a upozornění, která jsou pro uživatele aplikace zásadní, ale také informace o rezervacích přihlášeného uživatele. Tento informační panel bude rozdělen do tří částí.

První část obsahuje informace o rezervacích uživatele (Obr. 15).

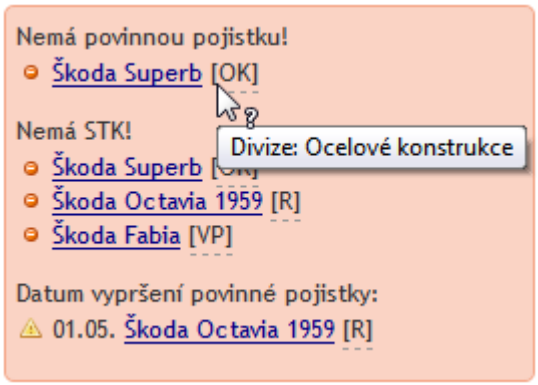
Obr 15: Příklad výpisu informačního panelu pro rezervace vozidel

V rámci ulehčení práce se systémem jsou ve všech informačních panelech zobrazovány ikonky a informace, nad kterými se po najetí myší zobrazí podrobnější popis (Obr. 16). Uživatel se tedy nemusí dále „proklikávat“ aplikací, aby zjistil přesnější informace o položce, zobrazené v tomto panelu.



Obr 16: Příklad výpisu podrobnějšího popisu informací v informačním panelu

Druhá část obsahuje varovný výpis pro stav pojistek a STK (Obr. 17).



Obr 17: Příklad výpisu informačního panelu pro STK a pojistky

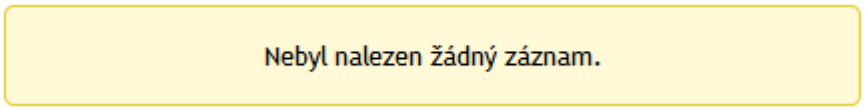
Třetí část obsahuje výpis neopravených závad, právě vykonávaných prohlídek a servisů a upozornění na nekompletní výbavu vozidel (Obr. 18).

Obr 18: Příklad výpisu informačního panelu pro doplňující informace o vozidlech

5.9 *Vytvoření informačních hlášení a varování*

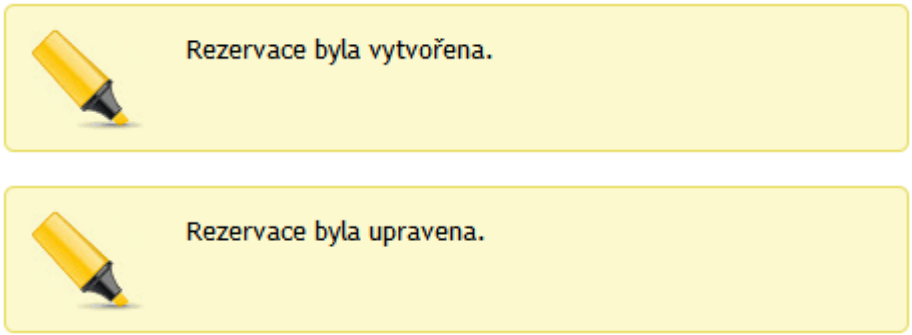
Informační hlášení a varování slouží k tomu, aby uživatel dostával potvrzení o tom, jakou operaci právě provedl nebo byl upozorněn před provedením zásadních operací. V aplikaci společnosti BKB Metal a.s. jsou použity čtyři typy hlášení.

Informace o nenalezených položkách v aplikaci (Obr. 19)



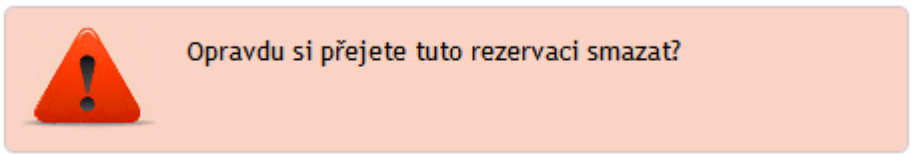
Obr 19: Příklad informačního hlášení výpisu prázdných položek v aplikaci

Informace o přidání nebo úpravě položky (Obr. 20)



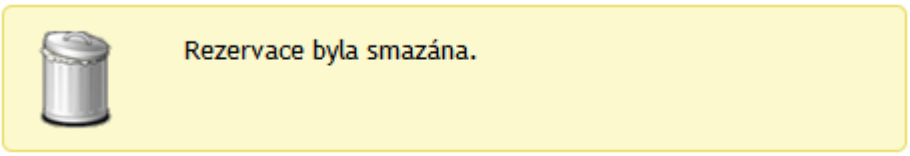
Obr 20: Příklad informačního hlášení pro přidání nebo úpravu položek v aplikaci

Varování před smazáním položky (Obr. 21)



Obr 21: Příklad varování před smazáním položky v aplikaci

Informace o provedeném smazání položky (Obr. 22).



Obr 22: Příklad informačního hlášení o provedeném mazání položky v aplikaci

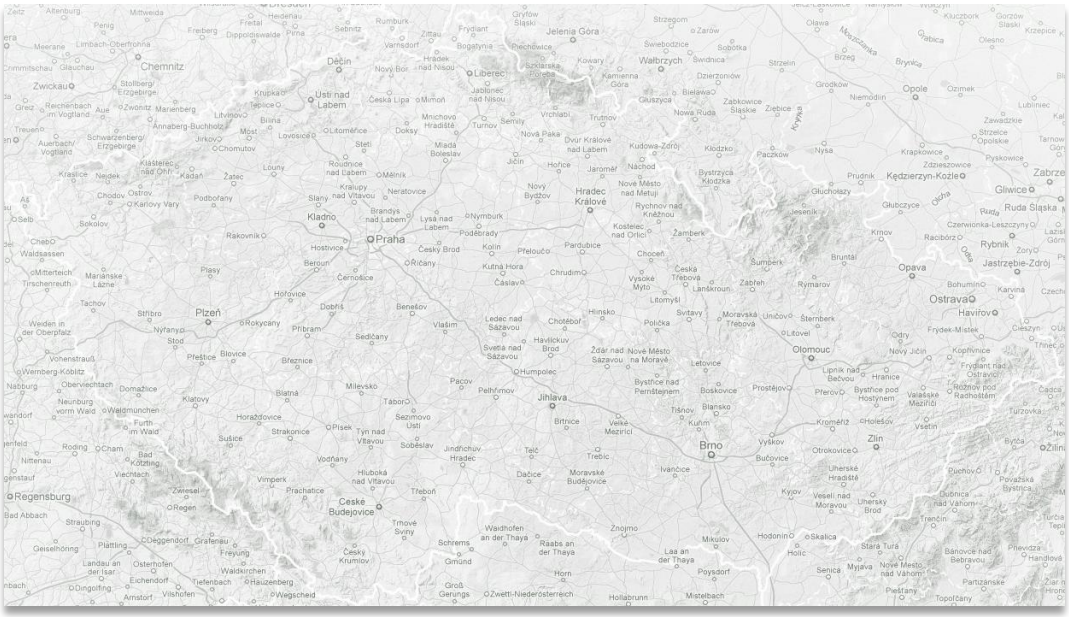
5.10 Design

Celkový vzhled aplikace je velice důležitý. Neměly by se používat výrazné barvy a musí být zvoleny vhodné přechody mezi barvami. Také styl písma je důležitý, jelikož některé styly písma mohou být špatně čitelné.

Pro design aplikace firmy BKB Metal a.s. byly použity světlé odstíny barev pro pozadí (především bílá) a sytější barvy pro zobrazení důležitých informací.

5.10.1 Pozadí

Pozadí bylo zvoleno tematicky jako mapa pro znázornění cestování vozidlem (Obr. 23). Zdrojem obrázku byla mapa na Google.cz Obrázek má po stranách světlé přechody a postupně přechází do bílé barvy.



Obr 23: Výřez z pozadí aplikace

5.10.2 Ikony

Pro přehlednější orientaci je v systému využívána sada malých ikon [4], které znázorňují, co bude která operace provádět nebo co která informace představuje. Funkce těchto ikon (Obr. 24) je také popsána v kapitole 5.8, kde je názorně ukázáno, co se zobrazí po najetí myší na jednu z ikon.

Obr 24: Příklad použitých ikon v aplikaci

5.11 Dokumentace

Dokumentace je velice důležitou součástí aplikace. Existují různé druhy dokumentací: Dokumentace ke specifikaci zadání, Dokumentace k analýze systému, Dokumentace k návrhu implementace, Uživatelská příručka a Programátorská příručka. Dokumentace se dodávají většinou se samotnou aplikací.

K aplikaci pro firmu BKB Metal a.s. byly vytvořeny tyto dokumentace: Uživatelská příručka a Programátorská příručka.

5.11.1 Programátorská příručka

Jedná se o nejdůležitější dokumentaci, protože je v ní popsána specifikace celé aplikace (je to jakési „knowhow“ aplikace). V případě, kdy na rozvoji aplikace bude pracovat někdo jiný než samotný autor, se právě využívá tato dokumentace. Z této dokumentace nově přidělený programátor snadno pochopí, jak je celá aplikace vytvořena, co obsahuje a jak byly jednotlivé prvky navrhnuty.

Programátorská příručka pro aplikaci správy vozového parku firmy BKB Metal a.s. obsahuje všechny potřebné informace, které vedou k sestavení celé aplikace. Každý prvek je podrobně popsán a všechny funkce v aplikaci jsou přesně definovány pomocí minispecifikací, o kterých pojednává kapitola 4.2.3. Celá programátorská příručka je součástí práce (Příloha A). Zde je pro úplnost pouze obsah programátorské příručky:

5.11.2 Uživatelská příručka

Uživatelská příručka slouží jako manuál pro koncového uživatele aplikace. Musí obsahovat popis všech operací pro jednotlivé role uživatelů v aplikaci. Z toho plyne, že každá role bude mít svou vlastní specifickou část v uživatelské příručce. Někdy se vytvářejí příručky zvlášť pro jednotlivé uživatelské role v aplikaci. Často bývá umístěna přímo v aplikaci jako forma nápovědy. Celá uživatelská příručka je součástí práce (Příloha A). Stěžejní prvky této dokumentace nejčastěji jsou:

- Na jakém hardware a software je aplikace provozuschopná
- Popis přihlášení do systému
- Popis jednotlivých částí menu pokud je potřeba
- Jaké jsou práva pro jednotlivé role v aplikaci
- Popis každé funkce v aplikaci
- Odpovědi na často kladené otázky

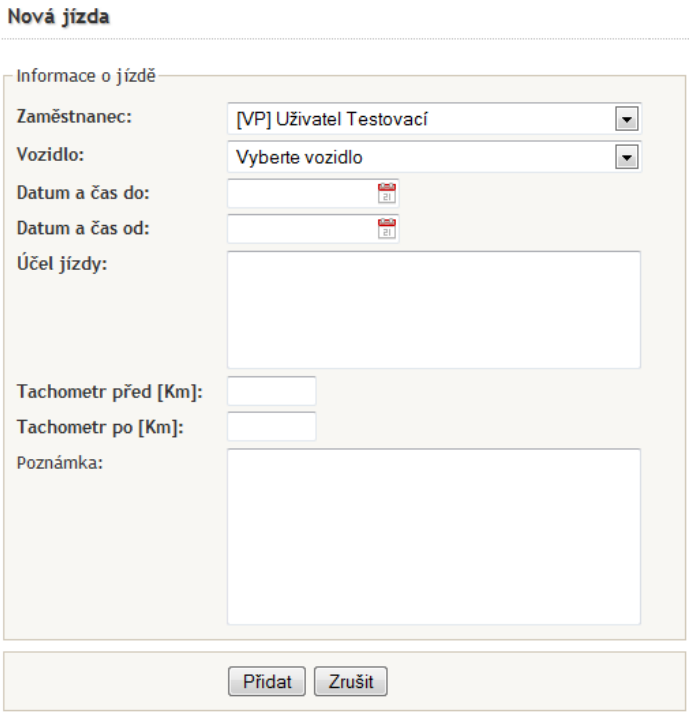
5.11.2.1 Výpis jízd pro roli „User“

Výpis knihy jízd (Orb. 25) pro roli „User“. V levém horním rohu je zobrazen titulek stránky. V pravém horním rohu je vyhrazeno místo pro vybrané ovládací prvky zobrazené stránky.

Obr 25: Příklad výpisu knihy jízd pro roli „User“

5.11.2.2 Vytvoření jízdy

Při vytváření nové jízdy (Obr. 26) uživatel vybírá zaměstnance (přednastaven je právě přihlášený zaměstnanec) a vozidlo z roletových menu.



Obr 26: Příklad vytvoření nové jízdy

Pro výběr data a času nové jízdy je uživateli zobrazen kalendář (Orb. 27):

Obr 27: Komponenta pro vkládání data a času

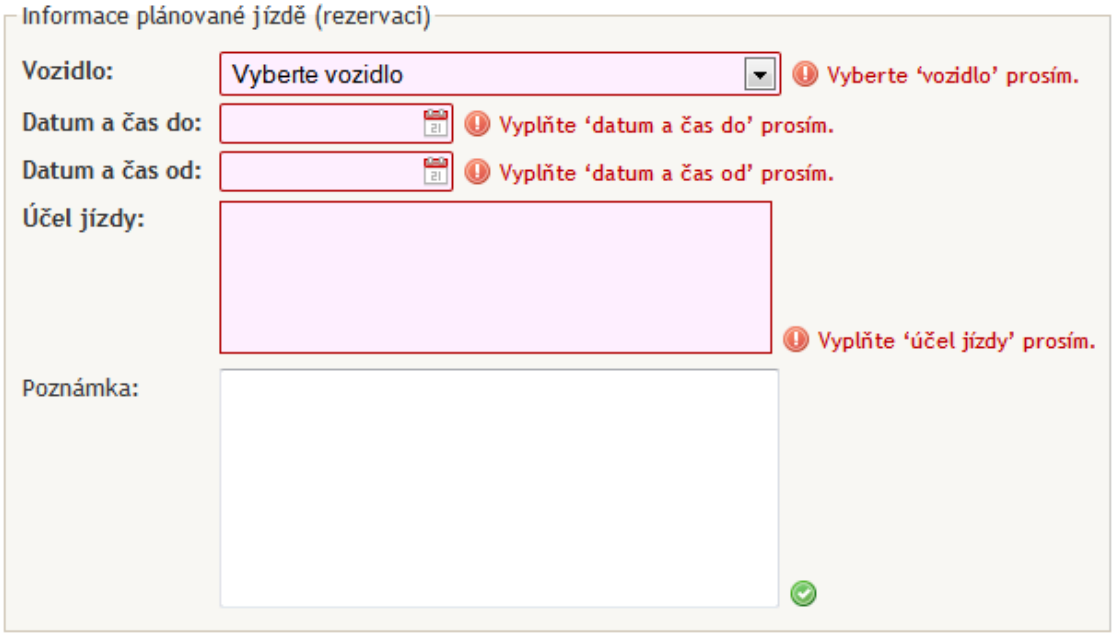
6. Zhodnocení funkčnosti aplikace

Celá aplikace musí být plně funkční, což znamená, že všechny funkce musí být proveditelné. Je velice důležité, aby aplikace sama kontrolovala správnost prováděných funkcí uživatelem. Jedná se například o validace dat vkládaných do aplikace. Dále musí aplikace sama vědět, co kterému uživateli zobrazovat podle rolí. Pod funkčnost také patří zabezpečení celé aplikace proti útokům a také zabezpečení přístupu. Funkčnost jednotlivých operací je ověřována testováním, které musí být prováděno postupně po celou dobu vývoje aplikace, aby nedocházelo ke vzniku chyb.

6.1 Validace dat v aplikaci

Velice důležitá je validace všech dat, které uživatel zadá do aplikace. Nebude tak docházet k tomu, že uživatel zadá nesprávná či nevalidní data. Tímto způsobem jsou ověřována či omezována data, která může uživatel zadat jako vstupy formulářových políček. Například pro vložení číselných hodnot musí být ověřeno, zda byla opravdu vložena číselná hodnota a nikoli textová. Někdy také potřebujeme specifikovat rozsah číselných hodnot, které uživatel může zadat.

Jako příklad si uvedeme formulář pro vytvoření rezervace (Obr. 28), kde jsou zobrazeny informace o tom, že nebyla vložena požadovaná data. Pouze pro vkládání poznámky je znázorněna správnost vstupu, protože poznámka je nepovinným údajem. Dokud nebude celý formulář správně vyplněn, nepustí aplikace uživatele dále. Rezervace tedy může vzniknout pouze na základě korektně vyplněného formuláře.



Obr 28: Validace vkládaných dat

6.2 Zabezpečení přístupu do aplikace

Každá aplikace by měla mít chráněný přístup a měli by se přihlásit pouze oprávnění uživatelé. V dnešní době je mnoho způsobů proniknutí do aplikace. Většina těchto útoků má za cíl získání citlivých dat uživatele (oběti útoku) či získání přístupu do administrační části aplikace za účelem napáchání škod. Popis a vysvětlení jednotlivých typů útoků však není náplní této práce. Proto si jen vyjmenujeme ty, se kterými se můžeme setkat nejčastěji. Jsou to například: *PHP Injection* [5], *SQL Injection* [6], *Cross-site scripting* (XSS) [7] nebo taky méně známé *Cross-Site RequestForgery* (XSRF) [8].

Všechny zmíněné a jiné typy útoků mohou být pro webové aplikace velmi nebezpečné. Tvůrce takové aplikace by tomu měl předcházet a udělat všechno proto, aby byly jeho aplikace proti těmto a jiným útokům zabezpečeny. Díky využití aplikačního rámce Nette a jeho komponent však na takováto zabezpečení myslet nemusíme, aplikaci ochrání za nás.

6.3 Omezení přístupu pro jednotlivé role

Celá aplikace je vytvořena tak, aby funkce určené specificky pro jednotlivé role mohli používat pouze a jen uživatelé, kteří mají tuto roli. Některé funkce (například hlasování v anketě) mohou využívat uživatelé všech rolí. Omezení přístupu spočívá v tom, aby se uživatelé s určitou rolí nikdy nedostali k funkcím, na které nemají oprávnění a neměli možnost takovouto funkci provést.

Aplikace pro správu vozového parku firmy BKB Metal a.s. nezobrazuje informace o tom, že právě tu či onu funkci smí či nesmí uživatel využít, ale automaticky zobrazuje pouze ty funkce, které náleží jeho roli.

6.4 Testování funkcí

Všechny funkce v aplikaci musí být proveditelné, tedy funkční a musí dělat právě to, k čemu jsou určeny. Testování funkcí je velice důležitým prvkem v celém průběhu tvorby aplikace.

6.5 Celkové zhodnocení funkčnosti

Aplikace byla vytvořena přesně podle požadavků firmy BKB Metal a.s. Všechny části zadání byly dodrženy a díky důkladně zpracované analýze a návrhu nebude docházet ke složitému přepracování celé aplikace. Aplikace má všechny vstupy plně validní, proto by k chybám v aplikaci nemělo docházet. Všechny operace a funkce v aplikaci fungují a jsou rozděleny přesně podle rolí. Testování aplikace probíhalo během celého vývoje a také bylo důkladně a úspěšně provedeno po dokončení. Aplikace je nyní připravena pro nasazení na firemní server.

7. PHP

PHP (Hypertext Preprocessor, dříve Personal Home Page) [9] (Obr. 29) je skriptovací programovací jazyk určený především k programování dynamických internetových stránek. Začleňuje se do struktury jiných jazyků, jako například HTML nebo XHTML, která je výhodná zvláště při tvorbě webových aplikací. PHP lze ovšem použít také ke tvorbě konzolových či desktopových aplikací.
PHP skripty jsou vykonávány na straně serveru a k uživateli je přenášen pouze výsledek jejich činnosti. Syntaxe jazyka kombinuje hned několik programovacích jazyků (Perl, C, Pascal a Java). Jazyk PHP je nezávislý na platformě a skripty fungují bez větších úprav na mnoha operačních systémech. Jeho základní vlastností je podpora mnoha knihoven, např. zpracování grafiky, textu, práce se soubory, přístup k většině databázových systémů a podporuje celé řady internetových protokolů. PHP je velmi oblíben především pro svou jednoduchost a širokou použitelnost. Jeho největší využití spočívá ve spojení s webovým serverem Apache a databázovým serverem MySQL pro tvorbu internetových aplikací.

Obr 29: Oficiální logo jazyka PHP

7.1 Obecné vlastnosti

Základní vlastností PHP jazyka je, že se jedná o dynamicky typový jazyk. Znamená to, že se datový typ proměnné určí v okamžiku přiřazení hodnoty. Jedná se o open source, tedy je volně dostupný. Jako každý jazyk má své výhody a nevýhody:

Výhody

- PHP je jazyk specializovaný na webové stránky
- Nativní podpora mnoha databázových systémů
- Obrovské množství projektů a kódů, které lze zdarma využít
- Poměrně jednoduchý na naučení
- Obsahuje rozsáhlý soubor funkcí v základní knihovně PHP
- Multiplatformnost (především Linux a Windows)
- Obrovská podpora na hostingových službách

Nevýhody

- Ve standardní distribuci chybí ladící (debugovací) nástroj
- Po zpracování požadavku neudržuje kontext aplikace, vytváří jej vždy znovu (oslabuje výkon)
- Útoky na aplikace psané v PHP s názvem *PHP Injection* [5]
- Nekonzistentní pojmenování funkcí
- Často nejednotné pořadí parametrů
- Ač jazyk podporuje výjimky, jeho knihovna je používá jen zřídka

7.2 Současnost PHP

V dnešní době se programátoři začali ubírat směrem k aplikačním rámcům neboli frameworkům, které vycházejí z tohoto jazyka, kde je jejich struktura rozdělena na jednotlivé menší prvky. Usnadňují celou práci, zajišťují větší přehlednost v kódu a hlavně samotné programování je rychlejší. Umožňují znovupoužitelnost již vytvořených částí aplikace. Tímto tématem se zabývá kapitola 8.

7.3 Ukázky kódu

Pro lepší představivost, jak se píše kód v jazyce PHP, je zde uvedena malá ukázka:

```php
<?php
// Zde je v proměnné string (tečka je operátor spojování řetězců)
$retez = "Ahoj, světe" . ', mám se dobře' . " a nevadí, že střídám oddělovače";

// Zde je v proměnné číslo (int)
$cislo = 100;

// Do proměnné je možné dát pole, které obsahuje jak čísla, tak znaky či další pole
$pole = array('a', 'b', 1, 2, array('první' => 'podpole', 'vytištěno'));

// Nenahlásí chybu (jenom varování) a vytiskne 'Array'
print($pole);

// Vytiskne obsah proměnné pole
print_r($pole);

// Test porovnání
$cislo = 100;
$retez = '100';

// Toto porovnání ('==') platí díky automatické typové konverzi
if ($retez == $cislo)   {
  echo 'Jsou stejné';
}

// Ale porovnání pomocí '===' neplatí, neboť nejsou stejné typy
if ($retez === $cislo)  {
  echo 'Jsou stejné';

} else {
  echo 'To by nešlo';
}
?>
```

7.4 Příklad webu vytvořeného pomocí PHP

Velké množství významných projektů, které jsou umístěny na internetu, byly implementovány právě za pomocí jazyka PHP. Určitě všichni známe největší světovou encyklopedii Wikipedia [10] (Orb. 30), která byla implementována v jazyce PHP.

Obr 30: Náhled na světovou internetovou encyklopedii Wikipedia

8. Nette a porovnání s ostatními aplikačními rámci

Jelikož psaní aplikací pouze v čisté formě jazyka může být u rozsáhlejších projektů zdlouhavé a náročné, proto pro implementaci aplikace vozového parku pro firmu BKB Metal a.s. byl zvolen PHP aplikační rámec. Tato práce je zaměřena také na porovnání zvoleného aplikačního rámce (dále frameworku) Nette s ostatními nejpoužívanějšími.

8.1 Framework obecně

Framework je softwarová struktura, která slouží jako podpora při programování a vývoji softwarových projektů. Může obsahovat podpůrné programy, knihovny tříd a funkcí neboli také API (Application Programming Interface) či doporučené postupy pro vývoj.

8.1.1 Účel frameworku

Hlavním cílem frameworku je převzetí typických problému, čímž se usnadní vývoj. Návrháři a vývojáři se pak mohou soustředit pouze na své zadání. Existují námitky, že použitím frameworku je kód pomalý či neefektivní, a že čas, který se ušetří použitím části již vytvořeného kódu, se musí věnovat nastudováním frameworku. Nicméně je zde obrovská úspora času při opakovaném nasazení často se opakujícího kódu nebo při tvorbě velkých projektů.

8.1.2 Architektura

Frameworky se obecně skládají z tzv. *hot spots* a *frozen spots*. *Frozen spots* definují celkovou architekturu, její základní komponenty a vztahy mezi nimi. Hlavní vlastností těchto částí je, že se nemění při použití frameworku. Naproti tomu *hot spots* jsou komponenty, které spolu s kódem programátora vytvářejí zcela specifickou funkcionalitu, a proto jsou pokaždé jiné.

8.2 Nette framework

Nette Framework [11] (Obr. 31) je výkonný framework pro pohodlné a rychlé vytváření kvalitních a moderních webových aplikací v PHP 5. Eliminuje bezpečnostní rizika, podporuje AJAX (změna obsahu stránek bez jejich „znovunačítání“), DRY (neopakovat se), KISS (řešit vše jednoduše), MVC (oddělení od nezávislých vrstev) a znovupoužitelnost kódu.

Výše uvedený odstavec ve zkratce vystihuje to, co od něj můžeme očekávat. S pomocí Nette Frameworku lze vytvořit dokonale zabezpečené aplikace, které jsou snadno udržitelné a rozšiřitelné. Přichází s celou řadou technologických inovací, které bychom u jiných frameworků marně hledali. Dalo by se říci, že vychází také z nedostatků jiných frameworků, které se snaží řešit nebo inovovat. Prudce zvyšuje produktivitu vývojáře (programátora) webových aplikací.

Obr 31: Oficiální logo Nette frameworku

Nette Framework (dále NF) je poměrně mladý a je stále vyvíjen. Vznik je datován na rok 2004, ale teprve v roce 2008 byl uvolněn jako open source a zpřístupněn veřejnosti. NF vytvořil český autor David Grudl [12], který se rozhodl využít svých mnoholetých zkušeností při vývoji webových aplikací a z důvodu nevyhovujících vlastností jiných dostupných frameworků vytvořil framework vlastní. Kolem tohoto frameworku vyrostla také jedna z nejaktivnějších komunit [13] českých PHP vývojářů komunikujících a sdílejících své zkušenosti prostřednictvím oficiálního fóra [14]. Tento fakt může být také jedním z důsledku toho, že komunitu vede český autor frameworku. Pomocí této komunity může do frameworku rychle proniknout i méně zkušený programátor.

Mladost frameworku má však také i své stinné stránky. V důsledku toho, že je framework stále ve vývoji, není kompletní dokumentace a v některých místech chybí dostatečně podrobné komentáře. Také v důsledku toho, že se od první verze frameworku některé části méně či více změnily a byly od té doby vydány verze nové, vznikl v nedávné době požadavek na přepsání Quick Startu [15], který už pár měsíců není ve finální podobě. Tento fakt by ale díky výše zmíněné komunitě neměl být velkou překážkou. Na komunitním fóru jsou velice přehledně popsané ukázky, díky kterým i méně zkušený uživatel začne chápat, jak s tímto frameworkem pracovat. V případě, že by uživatel na fóru nenalezl požadované vlákno, má možnost po registraci založit vlastní vlákno a zeptat se na věci, které ho zajímají. Odpovědi uživatelů fóra bývají někdy až překvapivě rychlé.

8.2.1 Quick Start

Quick Start (dále QS) by měl být jakýmsi prvním průvodcem vytvoření jednoduché aplikace v NF pro úplného začátečníka a také lákadlo pro ty, kteří nejsou ještě rozhodnutí jaký framework si vybrat. QS by měl být snadno pochopitelný, měl by čtenáři poskytnout základní informace, návyky a postupy při vývoji aplikace. Po jeho přečtení by měl čtenář umět vytvářet základní aplikace v NF. Naopak by QS neměl být dlouhý, nepřehledný a neměl by čtenáře zahltit přílišnými detaily.

Autor frameworku David Grudl také pořádá dvoudenní školení [16]. Je také možno shlédnout prezentaci o NF na jedné z jeho přednášek, které se konaly například na WebExpu 2009 nebo v listopadu 2009 v budově CPIT na VŠB-TUO [17]. Také je možné účastnit se neformálního setkání přátel NF pod názvem Poslední sobota [18] konaného vždy poslední sobotu v měsíci.

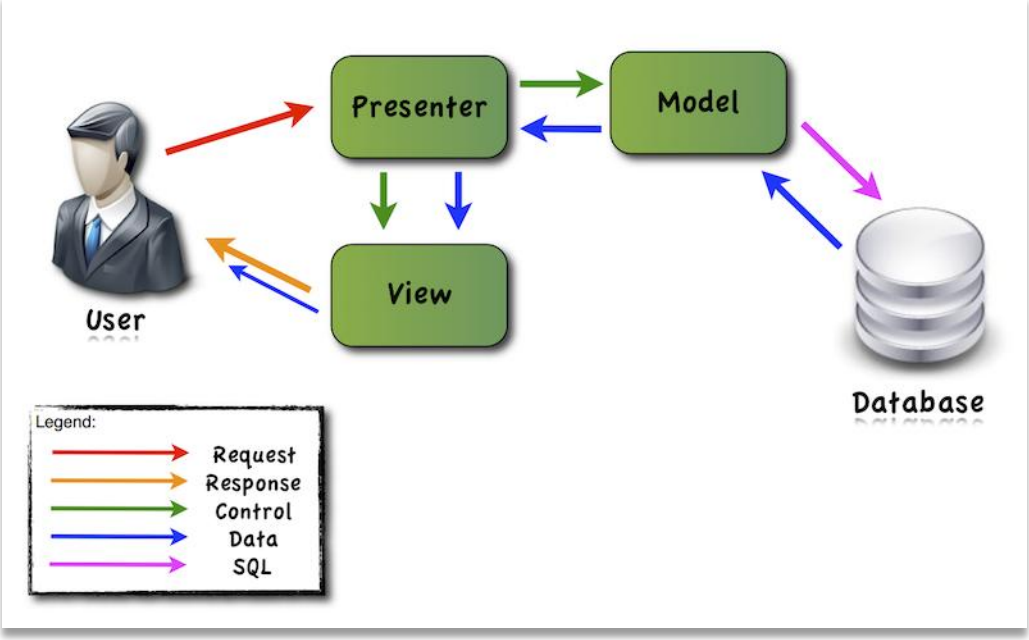
8.2.2 Obecné vlastnosti

Nette Framework je napsán promyšleným a čistým návrhem Objektově orientovaného programování (OOP) využívající nových vlastností PHP 5. Dle nezávislého testu [19] je Nette Framework jedním z nejrychlejších frameworků vůbec. Uživatele nijak neomezuje. Dokonce jej lze výhodně kombinovat například se Zend Frameworkem o kterém popisuje kapitola 8.3. Licence [20] patří k těm nejvolnějším. Framework tak lze zdarma používat i v komerčních projektech. Lze využít stále se rozšiřující nabídku pluginů [21] a ocenit tak možnosti znovu použitelnosti kódu.

8.2.3 Architektura

Využívá návrhový vzor *Model-View-Controller* (MVC) [22], tedy přesněji řečeno *Model-View-Presenter* (MVP) [23], jelikož Nette místo „controlleru“ využívá „presenter“. Tento návrhový vzor je ve skutečnosti velice jednoduchý a rozděluje aplikaci do 3 částí (Model, View, Presenter). Tyto části se mohou vyvíjet a hlavně udržovat samostatně. Tímto získává celý návrh na flexibilitě, přehlednosti a velkým způsobem zvýhodňuje práci ve skupinách.

Na obrázku (Obr. 32) je znázorněno, jakým způsobem tento návrhový vzor pracuje.



Obr 32: Schéma práce návrhového vzoru MVP

Popis jednotlivých částí MVP:

- **Model** – zajišťuje přístup k datům a manipulaci s nimi
- **View** – uvádí data reprezentovaná modelem po podoby vhodné pro prezentaci uživateli
- **Presenter –** reaguje na události přicházející od uživatele a zajišťuje změny v modelu nebo ve view

Laicky lze říci:

- **Model** by neměl vědět o tom, že existuje nějaký view či presenter
- **View** o modelu také vědět nemusí, nebo naopak může brát data přímo z něj
- **Presenter** má za úkol seznámit view s modelem (pozor, ne naopak) a realizuje uživatelské akce

Nette Framework (NF) striktně nediktuje programátorovi, aby v Modelu využíval nějaké konkrétní *ORM (Object-relational mapping)* [24], které mu nemusí vyhovovat. *ORM* je metoda sloužící k mapování relační databáze na objekty. NF žádné *ORM* nemá, což mohou někteří považovat za nedostatek. Místo toho NF dává uživateli naprostou volnost v tom, aby mohl použít nějaké již existující *ORM*, které zakomponuje do své aplikace, nebo může využít databázovou knihovnu *dibi* [25]. Ta byla rovněž vyvinuta autorem NF Davidem Grudlem a je s NF plně kompatibilní. V dnešní době ovšem již existuje *Ormion* [26] – *ORM* pro Nette Framework a *dibi* od Jana Marka [27], které je implementováno pro PHP 5.3 a momentálně běží na vývojové verzi NF. Je tedy dosti pravděpodobné, že se uživatelé NF dočkají i „vlastního“ *ORM* přímo pro Nette Framework.

Knihovna dibi je stejně jako Nette Framework stále ve vývoji a momentálně již podporuje značné množství významných databází, jako jsou: MySQL, PostgreSQL, SQLite, MS SQL, Oracle, Access, PDO a ODBC. Maximálně ulehčuje práci programátorům tím, že zjednodušuje zápis SQL příkazů. Lze použít snadný přístup k metodám i bez globálních proměnných a obsahuje funkce pro několik rutinních úkonů (SELECT, UPDATE, DELETE atd.). Dále eliminuje chyby přehledným zápisem SQL příkazů. Podporuje také přenositelnost mezi databázovými systémy a podporuje konvence pro různé typy databází. Automaticky formátuje speciální datové typy (například datum, řetězec) a sjednocuje základní funkce jako připojení k databázi, vykonání příkazu a získání výsledku. Hlavně zachovává maximální jednoduchost.

8.2.4 Důležité prvky v Nette

Nespornou výhodou NF, je tzv. „Skeleton". Jedná se o kostru, tady jakousi základní aplikaci, kterou může programátor využít při tvorbě nové aplikace. Tuto základní aplikaci můžeme nalézt přímo v distribučním balíku Nette frameworku. Skeleton programátorovi nové aplikace nabízí (nikoli však vnucuje) základní adresářovou a souborovou strukturu. Z toho vyplývá, že si může uživatel do jisté

míry sám zvolit, jakou strukturu bude jeho aplikace mít. V rámci přehlednosti a zachování standardu je doporučováno, aby uživatel vycházel z této zadané adresářové struktury.

Routování

V dnešní době vyžadují moderní dynamické webové aplikace sbírat požadavky zvenčí a na jejich základě přistupovat ke zdrojům dat, které poté patřičně „odprezentují“. Aby aplikace věděla jaké akce má vykonávat při různých požadavcích, musí se nejprve určit nějaká pravidla. Této technice se říká *routování* a typickým požadavkem zvenčí je pro webové aplikace URL adresa.

U webových aplikací založených na architektuře (neboli vzoru) MVC a MVP je *routování* (neboli směrování) prostředníkem právě mezi požadavkem zvenčí a správným řadičem, který požadavek zpracuje a zobrazí očekávaný výstup.

Právě tvar „routy“ určuje, kterému presenteru a pohledu (view) bude požadavek přidělen a jak bude zpracován. V současné době se setkáváme s „lidštějšími URL“, které jsou použitelnější a zapamatovatelnější než jejich „syroví“ předchůdci. Daleko větší důležitost ale sehrávají v současně aktuálních optimalizacích webů pro vyhledávače (*Search Engine Optimalization* – *SEO* [28]). Nette Framework na současné trendy myslí a vychází v tomto vývojářům plně vstříc.

Příklad nastavení routování v Nette Frameworku:

```
$application = Environment::getApplication();
$router = $application->getRouter();

$router[] = new Route('index.php', array(
        'module'        => 'Admin',
        'presenter' => 'Book',
), Route::ONE_WAY);

$router[] = new Route('<presenter>/[page-<page>/]<action>/<id>', array(
        'module'        => 'Admin',
        'presenter'     => 'Book',
        'action'        => 'default',
        'id'            => null,
        'page'          => 1,
));
```

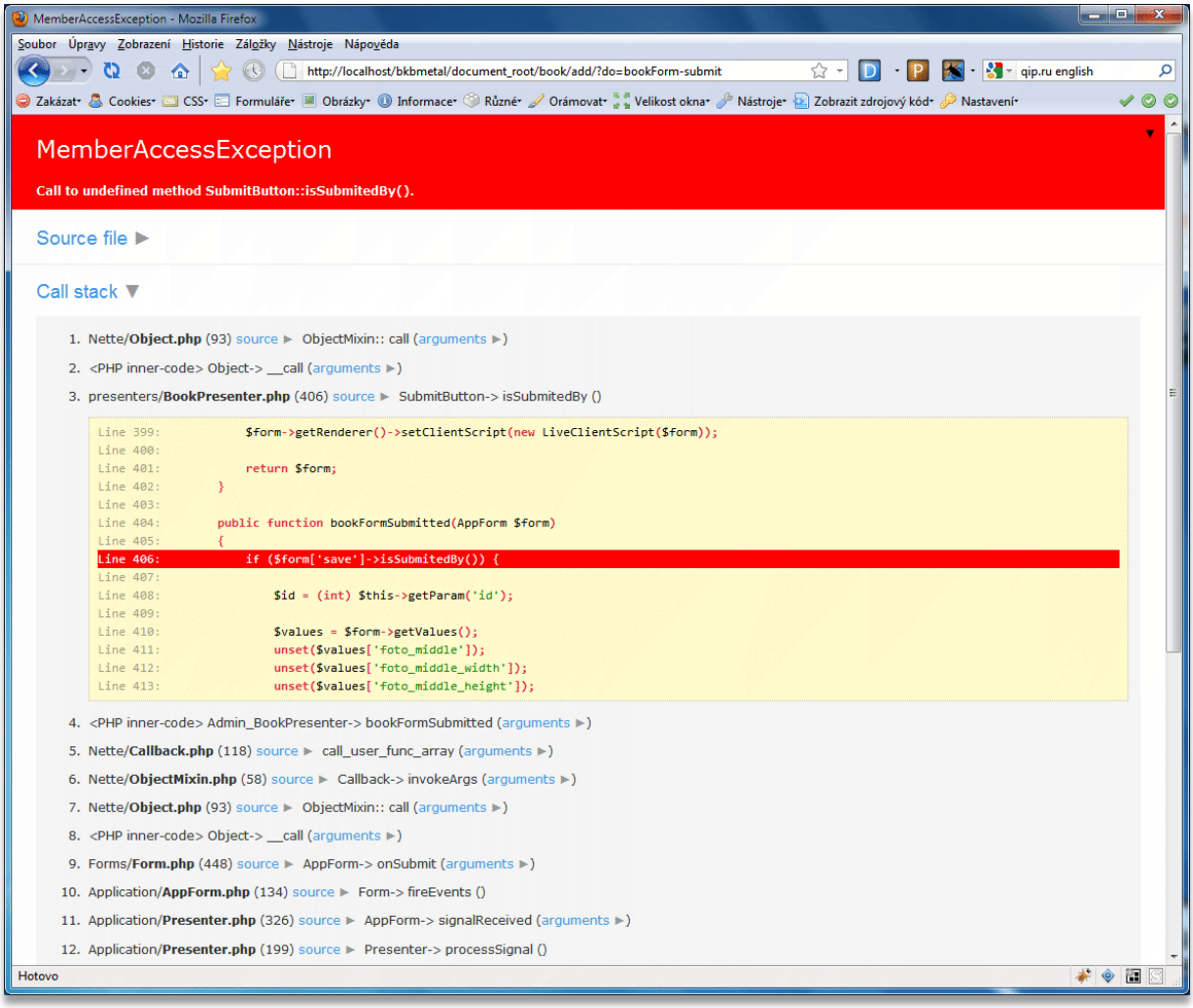
Obousměrný přepis URL

Přepis URL na akce je základem většiny webových frameworků. V takových systémech má routování obrovský význam. Naopak v Nette se jím není třeba zabývat. Je to věc, kterou programátor může řešit, až když je aplikace hotová. Navíc není vůbec žádný problém kdykoliv úplně změnit veškeré cesty pouze přepsáním „rout“, protože routování je obousměrné a slouží ke generování URL cest. To dovoluje strukturu URL navrhovat nezávisle na zbytku aplikace.

Laděnka

Velkým přínosem pro vývoj aplikací v Nette Frameworku jsou přehledně zpracovaná chybová hlášení (Obr. 33), neboli Laděnka. Vypisovaní chyb se však nikdy nesmí dostat na produkční server. Pokud ji programátor používá na vývojovém serveru, je vše v pořádku. Pokud by ale došlo k zobrazení Laděnky na veřejném (produkčním) serveru, mohla by vyzradit hodně citlivých informací (nicméně kdyby k tomu náhodou došlo, má integrovaný systém pro skrytí citlivých políček, např. hesel). Na produkčním serveru je však možné nechat chybová hlášení ukládat do adresáře nebo odesílat administrátorovi emailem.

Laděnka tedy funguje buď v režimu zobrazování, nebo logování. Režim laděnky je možno nastavit parametrem nebo pokud jej neuvedeme, detekuje se režim na základě IP adresy serveru. Je-li na adrese veřejné, půjde o produkční režim, je-li na lokální, tak o „vývojářský“ režim.



Obr 33: Příklad výpisu chybového hlášení pomocí Laděnky

Autoloading tříd

RobotLoader v Nette zaručuje velmi rychlé a samostatné načítání tříd, což samo o sobě zrychluje celou práci. Obsahuje vlastní obsluhu pro načítání tříd. Jejím jádrem je vcelku jednoduchá funkce, která v adresářích a podadresářích webové aplikace projde všechny PHP skripty a vyhledá v nich definice tříd a rozhraní. Výsledkem je tabulka identifikátorů a relativních cest k souborům. NF pak přesně ví, který soubor při požadavku na konkrétní třídu vložit. Tento proces je velice rychlý. Tabulka je uchovávána na disku, v podobě INI souboru.

K načtení NF do aplikace stačí zavolat pouze jediný příkaz. NF si už pak samo zjistí, jaké třídy má načíst.

```
// absolutní cesta k rootu webu
define('WWW_DIR', dirname(__FILE__));

// absolutní cesta ke knihovnám
define('LIBS_DIR', WWW_DIR . '/../libs');

// tímto řekneme, aby Nette Framework načetl třídy automaticky,
// tedy nemusíme vyjmenovávat soubory příkazem 'require'
require LIBS_DIR . '/Nette/loader.php';
```

8.2.5 Ukázky kódu

Pro pochopení stylu vytváření aplikace za pomocí Nette Frameworku je zde ukázka kódu z jeho tří hlavních částí: model, view a presenter.

Ukázka kódu z modelu:

Jedná se o základní model, ve kterém je nastaveno připojení k databázi. Každý model, který bude potomkem tohoto modelu, bude mít automaticky také připojení k databázi a není tedy nutné jej definovat v každém modelu znova, což je názorným příkladem DRY (neopakovat se).

```
abstract class BaseModel extends Object
{
        private static $db;

        /**
        * Getter funkce zajistujici pripojeni k databazi pri vytvoreni modelu
        */
        public function getDb()
        {
                if (self::$db == NULL) {
                  self::$db = new DibiConnection(Environment::getConfig('database'));
                }
                return self::$db;
        }
}
```

Následující model využívá základního modelu a jeho připojení k databázi a ukazuje základní operace, které nad daty v databázi budeme provádět.

```
final class BookModel extends BaseModel
{
        private $table          = 'kniha_jizd';
        private $id_table       = 'id_jizda';

        public function find($id)
        {
                $result = $this->db->select('*')
                                   ->from($this->table)
                                   ->where($this->id_table . '=%i', $id);
                return ($result) ? $result : NULL;
        }

        public function insert(array $data)
        {
                $result = $this->db->insert($this->table, $data)
                                   ->execute(dibi::IDENTIFIER);
                return ($result) ? $result : NULL;
        }

        public function update($id, array $data)
        {
                $result = $this->db->update($this->table, $data)
                                   ->where($this->id_table . '=%i', $id)
                                   ->execute();
                return ($result) ? $result : NULL;
        }

        public function delete($id)
        {
                $result = $this->db->delete($this->table)
                                   ->where($this->id_table . '=%i', $id)
                                   ->execute();
                return ($result) ? $result : NULL;
        }
}
```

Ukázka kódu z presenteru:

V presenteru pracujeme v metodách *action* s modely a v metodách *render* s pohledy.

```
final class Admin_DivisionPresenter extends Admin_SecuredPresenter
{
        public function actionDefault()
        {
                $divisionModel = new DivisionModel();
                $this->template->divisions = $divisionModel->findAll()
                                                   ->orderBy('stav_divize', dibi::ASC)
                                                   ->orderBy('zkratka_divize', dibi::ASC)
                                                   ->fetchAll();
        }

        public function renderDefault()
        {
                $this->template->pageTitle = 'Divize';
        }
}
```

Ukázka kódu z view:

Šablony (neboli také pohledy) jsou v NF přehledné, snadno čitelné a dobře se píší.

```
{block #title}{$pageTitle}{/block}

{block #titleLinks}
<a href="{link add}">Nová divize</a>
{/block}

{block #content}

<table n:if="!empty($divisions)">
        <tr>
                <th>Název divize</th>
                <th>Zkratka</th>
                <th>Stav</th>
                <th> </th>
        </tr>
        <tr n:foreach="$divisions AS $division" rel="{$division->id_divize}">
                <td>{$division->nazev_divize}</td>
                <td>{$division->zkratka_divize}</td>
                <td>{$division->stav_divize}</td>
                <td><a href="{link edit, $division->id_divize}">upravit</a></td>
        </tr>
</table>

<div n:if="empty($divisions)">Nebyl nalezen žádný záznam k zobrazení.</div>
```

8.2.6 Šablonovací systém

Jedná se o systém, který slouží k vytváření šablon a formulářů a díky jeho vlastnostem je práce v NF mnohem jednodušší. Řeší například zabezpečení proti útokům na aplikace. Jedním z nejtriviálnějších a nejčastějších způsobů narušení webových stránek je *Cross Site Scripting* (XSS) [7]. Ať už z pohledu principu útoku nebo obrany proti němu, přesto jde dost možná o nejčastější zranitelnost. Obranou je „escapování“, tedy převod znaků majících v daném kontextu speciální význam na jiné odpovídající sekvence.

Ve webových prezentacích se nejčastěji setkáváme s pěti kontexty: HTML, JavaScript, URL, CSS a XML. Funkce escapující HTML by měla patřit k nejčastěji volaným funkcím PHP, proto ji autoři dali krátké a už od pohledu srozumitelné jméno *htmlspecialchars* [29]. V každém kontextu se ale *escapuje* jinak. Proto NF přichází s revoluční technologií *Context-aware escaping* [30], která rozpozná kontext a automaticky zvolí správné *escapování*.

Jak už bylo řečeno, proměnné v šablonovacím systému NF se *escapují* automaticky. Pokud bychom si ovšem nepřáli, aby byl obsah proměnné *escapován* (například pokud by proměnná obsahovala HTML kód) a chtěli bychom ji vypsat bez jakékoliv transformace, stačí přidat vykřičník:

```
...
<div>{!$row->html_code}</div>
...
```

Setkáváme se tak s principem, kdy méně psaní vede k většímu zabezpečenému kódu. Pokud bychom chtěli vypnout escapování, tak stačí přidat vykřičník. Nebo naopak, pokud vykřičník zapomeneme napsat, nedopustíme se bezpečnostní chyby.

Syntaxe šablonovacího systému, která byla předložena v ukázce kódu z view, může být povědomá. Podobnou používá řada šablonovacích systémů. Z těch nejznámějších například *Smarty* [31].

8.2.7 Editor pro práci v Nette

NF svůj vlastní editor nemá. Lze ale použít jiné *IDE* (v překladu: vývojové prostředí) editory jako například *NuSphere PhpED*, ve kterém funguje jak debugování tak našeptávání kódu. Tento *IDE* editor je zajímavý vlastním debugovacím toolbarem [32], který si programátor může doinstalovat jako plugin do prohlížeče *Firefox* [33]. Mezi další často využívaná rozšíření Firefoxu patří take *Firebug* [34] s doplňkem *FirePHP* [35] s nímž umí Nette Framework automaticky komunikovat. Dalšími *IDE* editory, které jsou také využívány, jsou například *NetBeans* [36] či *Eclipse* [37].

8.2.8 Příklad webů vytvořených v Nette frameworku

NF je v dnešní době stále populárnější a pohání již spoustu webových prezentací či složitých webových aplikací. Jako příklad si uvedeme webové stránky našeho prezidenta Václava Klause [38], nebo populární server *Root.cz* [39].

8.2.9 Zhodnocení

Jedná se o jeden z nejočekávanějších frameworků vůbec. Zatím velkou slabinou se zdá být neúplná dokumentace, což můžeme přičíst mladosti frameworku a také tomu, že se o vše stará v podstatě jeden člověk. Je možné, že kdyby autor povolil podílení se na vývoji frameworku i dalším zkušeným PHP programátorům, mohl by být vývoj celkově rychlejší. Na druhou stranu má tak autor plnou kontrolu nad celým dílem a vývoj frameworku se ubírá pouze tím směrem, který si on sám určuje. Velkým plusem je možná kombinace se Zend frameworkem či jinými frameworky a knihovnami. Vyzdvihnout můžeme také velmi dobré vlastnosti routování, velmi úsporně vyřešené vytváření formulářů [40] a celý šablonovací systém. Podrženo a sečteno, Nette Framework je momentálně jeden z nejvyužívanějších frameworků i vzhledem k jeho mladosti. Dokazuje to také, že i v naší zemi jsou zdatní PHP programátoři a tento framework se stále více tlačí do světa.

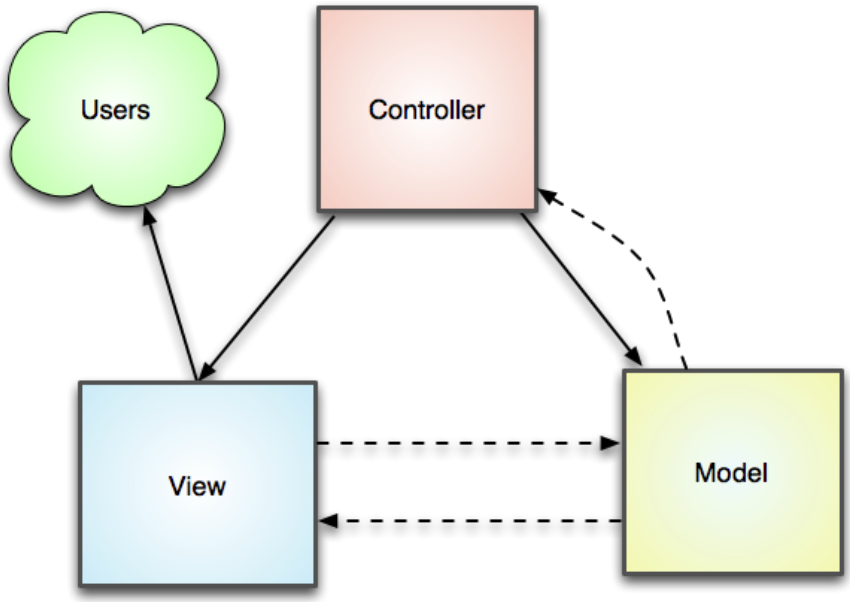
8.3 Zend framework

Zend framework [41] (Obr. 34) je vydán pod open source licencí stejně jako Nette Framework a využívá také návrhový vzor MVC (Model-View-Controller) [22]. Jeho vývoj je datován k roku 2005 a je implementován v PHP 5. Je velice hojně využíván a komunita kolem něj je obrovská.

Obr 34: Oficiální logo frameworku Zend

8.3.1 Obecné vlastnosti

Zend Framework (dále jen ZF) má velice přehledně zpracovanou dokumentaci, která je kompletně v angličtině, což je velkým plusem. Pokud člověk neumí moc anglicky, raději by se měl zaměřit na framework Nette, který je vyvíjen českou komunitou, tudíž pronikání do samotného frameworku bude mnohem rychlejší. Tím, že je celý framework postaven na architektuře MVC (Obr. 35) je zajištěno, že sám uživatel se v aplikacích vytvořených pomocí Zend Frameworku bude velice dobře orientovat a výsledný kód bude přehledný. Velice užitečnou vlastností je User-friendly URI, což znamená přehledný výstup URL adresy.



Obr 35: Schéma práce návrhového vzoru MVC

8.3.2 Ukázky kódu

Ukázka příkladů zdrojových kódu pomůže nahlédnout tak trochu pod pokličku vývoje aplikace v ZF. Zde jsou uvedeny tři ukázky nejdůležitějších částí frameworku, a to view, model a controller:

Ukázka kódu z view:

```
<table cellspacing = "10" border="1" cellpadding = "5">

        <?php foreach($this->useri as $user) : ?>
        <tr>
                <td><?php echo $this->escape($user->ID);?></td>
                <td>
                        <?php
                        echo $this->escape($user->name)
                        ." ".
                        $this->escape($user->surname);
                        ?>
                </td>
                <td><?php echo $this->escape($user->email);?></td>
                <td><?php echo $this->escape($user->phone);?></td>
                <td>
                        <a href="<?php
                                echo $this->baseUrl;
                                ?>/users/edit/id/<?php
                                echo $user->ID.'/id_clena/'.$user->id_member;
                                ?>">Edit</a>
                </td>
        </tr>
        <?php endforeach; ?>
</table><br />
<a href="<?php echo $this->baseUrl; ?>/index">Zpět na seznam členů</a>
```

Ukázka kódu z modelu:

```
<?php
class Users extends Zend_Db_Table
{
        protected $_name = 'users';
}
?>
```

Ukázka kódu z controlleru:

```php
<?php
require_once 'Zend/Controller/Action.php';
require_once 'Zend/Session/Namespace.php';
require_once 'Zend/Form.php';
require_once 'Zend/Form/Element/Text.php';

class UsersController extends Zend_Controller_Action
{
        function indexAction()
        {
                $useri = new Users();
                $id = (int)$this->_request->getParam('id', 0);
                $member = new Members();
                $this->view->member = $member->fetchRow('ID='.$id);
                $this->view->title = "Uživatelé člena ".$this->view->member->name;
                $namespace = new Zend_Session_Namespace();
                if (isset($namespace->vypis))
                {
                        $this->view->vypis = $namespace->vypis;
                        unset($namespace->vypis);
                }

                if ($id > 0) $this->view->useri = $useri->fetchAll('id_member='.$id);
        }
}
?>
```

8.3.3 Editor pro práci v ZF

Díky tomu, že ZF je velice rozšířen, má nejen vlastní editor pro psaní aplikací, ale i vlastní server (Obr. 36). Lze stáhnout z oficiálních stránek [42]. Tyto možnosti způsobují větší zájem o ZF.

Obr 36: Oficiální loga produktů Zend

8.3.4 Příklad webu vytvořeného pomocí ZF

Jako příklad stránek vytvořených v tomto frameworku si uvedeme server *Výsledky.cz* [43] a internetový obchod *Shopio* [44].

8.3.5 Zhodnocení

Ačkoliv je ZF velice populární a nabízí velice mnoho zajímavých možností, rychlost přístupu do databáze je velice pomalá. Což je bráno jako obrovská nevýhoda. Výsledný dojem pro pokročilého

uživatele ze ZF je velice příjemný, ale většina Čechů se momentálně přiklání k Nette, jelikož jeho časová návaznost na aplikace je mnohem lepší, má rozsáhlejší českou komunitu a výhledově do budoucna je určitě mnohem zajímavější.

8.4 CodeIgniter

CodeIgniter [45] (Obr. 37) je open source framework založený na architektonickém principu MVC [22] (Obr. 35). Je určen pro dynamický vývoj webových aplikací v PHP. CodeIgniter je vyvíjen americkou společností EllisLab [46] s rozsáhlou internetovou komunitou. Jeho první verze byla zveřejněna v únoru roku 2006. Jako všechny frameworky má za úkol ušetřit práci vývojářům aplikací. Je považován za jeden z nejjednodušších frameworků.

Obr 37: Oficiální logo frameworku CodeIngniter

8.4.1 Obecné vlastnosti

K provozování aplikace vytvořené za pomocí frameworku CI je potřeba běžný webový server. Samotný Framework je vyvíjen jako odlehčený. Znamená to tedy, že jádro systému závisí jen na několika malých knihovnách. Další potřebné knihovny jsou nahrávány až v případě potřeby a za běhu. Tento způsob dynamického nahrávání zdrojů velmi přispívá k rychlosti frameworku. Veškerá konfigurace probíhá editací různých souborů v adresáři application/config. Jedná se především o tato nastavení:

- routes.php – umožňuje přepisovat URI dotazy
- config.php – různá nastavení pro správný chod aplikace
- database.php – nastavení přístupu k databázi
- autoload.php – automatické nahrávání zdrojů jako jsou knihovny, modely a pomocné funkce (helpery)

Komunita okolo tohoto frameworku je hodně početná, až přes 100 000 aktivních přispívajících na oficiálním fóru. Tento framework má velice příkrou křivku učení práce díky velké komunitě a spoustě příkladů k procvičení. I když je komunita na fóru velice početná, tento framework je vyvíjen pouze firmou EllisLab. Hlavní nevýhodou je, že nepoužívá layouty.

8.4.2 Ukázky kódu

Pro lepší porovnání s jinými frameworky si na tomto místě ukážeme, jak se v CodeIgniter vytváří model, view a controller:

Ukázka kódu z modelu:

```php
<?php
class Function_model extends Model {
    function Function_model() {
        parent::Model();
    }

    function getSearchResults ($function_name, $description = TRUE) {
        $this->db->like('function_name', $function_name);
        $this->db->orderby('function_name');
        $query = $this->db->get('functions');
        if ($query->num_rows() > 0) {
            $output = '<ul>';
            foreach ($query->result() as $function_info) {
                if ($description) {
                    $output .= '<li><strong>'
                               . $function_info->function_name
                               . '</strong><br />';
                    $output .= $function_info->function_description . '</li>';
                } else {
                    $output .= '<li>' . $function_info->function_name . '</li>';
                }
            }
            $output .= '</ul>';
            return $output;
        } else {
            return '<p>Sorry, no results returned.</p>';
        }
    }
}
?>
```

Ukázka kódu z view:

```php
<?php
$this->load->view("header");
?>
    <h2><?= $title;?></h2>

    <div id="function_description">
        <?= $search_results;?>
    </div>

<?php
$this->load->view("footer");
?>
```

Ukázka kódu z controlleru:

```php
<?php
class Application extends Controller {

    function Application()
    {
        parent::Controller();
        $this->load->model('function_model');
        $this->output->cache(120);
    }

    function index()
    {
        $data['title'] = "Code Igniter Sample Application";
        $data['extraHeadContent'] = '<script type="text/javascript" src="'
                                    . base_url()
                                    . 'js/function_search.js"></script>';
        $this->load->view('application/index', $data);
    }

    function search()
    {
        $data['title'] = "Code Igniter Search Results";
        $function_name = $this->input->post('function_name');
        $data['search_results'] = $this->function_model
                                    ->getSearchResults($function_name);
        $this->load->view('application/search', $data);
    }
}
?>
```

8.4.3 Příklad webu vytvořeného pomocí CI

CodeIngniter není stvořen pro robustní aplikace (vysvětleno ve zhodnocení). Jako příklad si uvedeme tyto dvě webové prezentace: *Pacific Peoples' Partnership* [47] a *nuherbs co* [48].

8.4.4 Zhodnocení

Celkově shrnuto, CodeIgniter staví na jednoduchosti a rozšiřitelnosti. Řeší vývoj aplikací spíše klasickou PHP cestou, tedy bez komplexního OOP. Používá se především pro vytváření menších aplikací. Jelikož jej vyvíjí pouze firma EllisLab a komunita kolem něj nespolupracuje na výboji, je pro tento framework velice složité konkurovat jiným frameworkům.

8.5 CakePHP

Tento framework [49] (Obr. 38) vznikl v roce 2005 jako reakce na úspěch *Ruby on Rails* [50] (jeden z prvních frameworků vůbec). Je bohatě vybavený a je hodně řízen konvencemi, což oslabuje jeho variabilitu. Zaměřuje se především na přehledný a co nejkratší kód pro tvorbu aplikací. Velice často je kombinován se ZF, kde se využívá jednoduchost a přímočarost CakePHP a síla ZF.

Obr 38: Oficiální logo frameworku CakePHP

8.5.1 Obecné vlastnosti

Stejně jako výše zmíněné frameworky využívá návrhový vzor MVC [22] (Obr. 35), díky kterému má přehledně rozdělen celý kód. Dokáže definovat jak administrační tak normální akce v jednom controlleru, což je obrovská výhoda. Využívá podporu odděleného layoutu, kterou například nepodporoval CI. CakePHP je celkem snadný k naučení, jelikož má velké množství ukázkových příkladů. Na druhou stranu ale tento framework obsahuje celkem velké množství funkcí, které jsou špatně zpracované či nepoužitelné. Další velkou nevýhodou je nekompatibilita se staršími verzemi.

8.5.2 Ukázky kódu

Tato ukázka příkladů view, modelu a controlleru umožní částečné pochopení myšlenky vytváření aplikací v tomto frameworku.

Ukázka kódu z view:

```
<?php echo $this->renderElement('admin_main_menu'); ?>
<h1>Edit Post</h1>
<form method="post" action="<?php echo $html->url('/admin/posts/edit')?>">
    <?php echo $form->hidden('Post/id'); ?>

    <p>Published <?php echo $form->checkbox('Post/active'); ?></p>
    <p><?php echo $form->input('Post/title', array('size' => '50'))?></p>
    <p>Body: <?php echo $form->textarea('Post/body',
                                        array('rows'=>'30',
                                              'cols' => '80')) ?>
    </p>
    <p>Comments: <?php echo $form->radio('Post/discussion_type',
                                         array('1' => 'Open',
                                               '2' => 'Moderated',
                                               '3' => 'Closed'),
                                         null); ?>
    </p>
    <p><?php echo $form->submit('Save')?></p>
</form>
```

Ukázka kódu z modelu:

```
class Post extends AppModel
{
    var $name = 'Post';
    var $belongsTo = 'User';
    var $hasAndBelongsToMany = array("Tag");
    var $validate = array(
        'title'   => VALID_NOT_EMPTY,
        'body'    => VALID_NOT_EMPTY
    );

    function beforeSave()
    {
        if (empty($this->id)) {
           // New
           $this->data[$this->name]['url'] = $this->getUniqueUrl(-1,
                                             $this->data[$this->name]['title'],
                                             'url');
        } else {
           // Update
           $this->data[$this->name]['url'] =
              $this->getUniqueUrl($this->data[$this->name]['id'],
          $this->data[$this->name]['url'],'url');
        }

        if ($this->data[$this->name]['active'] == 1) {
            $this->data[$this->name]['published'] = date('Y-m-d H:i:s');
        }

        return true;
    }
}
?>
```

Ukázka kódu z controlleru:

```php
<?php
class PostsController extends AppController
{
	var $name = 'Posts';
	var $layout = 'default';

	function index()
	{
		$this->set('posts', $this->Post->findAll());
	}

	function view($url)
	{
		$post = $this->Post->findByUrl($url);
		$this->set('post', $post);
	}

	/* Admin akce */

	function admin_list()
	{
		$this->checkUser();
		$this->layout = 'admin';
		$this->set(
			'posts',
			$this->Post->findAll('WHERE `Post`.`user_id` =
				`'.$this->Session->read('User.id')
				.'` ORDER BY modified DESC,created DESC')
			);
	}

	function admin_new()
	{
		$this->layout = 'admin';
		$this->set('page', array('posts', 'new'));
		$this->set("tags", $this->Post->Tag->generateList());

		if (!empty($this->data))
		{
			$this->data['Post']['user_id'] =
				$this->Session->read('User.id');

			if ($this->Post->save($this->data))
			{
				$this->flash(
					'Your post has been saved.',
					'/admin/posts/list'
				);
			}
		}
	}
}
?>
```

8.5.3 Příklad webu vytvořeného pomocí CakePHP

Jako příklad webové aplikace vytvořeného v CakePHP si uvedeme rozsáhlou sbírku skriptů *HotScripts* [51] a stránky *Mark Story* [52].

8.5.4 Zhodnocení

Tento framework je zastíněn svými většími kolegy, tudíž není tak hojně využíván. Je ale velice doporučován jako prvotní volba pro uživatele, který nikdy ve frameworku nepracoval. Pro většinu uživatelů je ale tento krok zbytečný a raději začnou rovnou s nějakým mocnějším frameworkem.

8.6 Symfony

Jedná se o webový aplikační framework [53] pro vývoj webových aplikací vycházející z návrhového vzoru MVC [22] (Obr. 35). Vznikl původně pod názvem Sensio Framework. Vychází z co největší úspory psaní kódu programátorem. Dostal se tak daleko, že poměrně velkou část kódu „rozsekal“ do konfiguračních souborů, které se píšou ve velmi úsporném a čitelném syntaktickém zápisu zvaném YAML [54]. Je to deklarativní metoda programování, a pokud si na ni uživatel zvykne, je jistě velmi produktivní. Na druhou stranu je pravda, že ovládnout Symfony (Obr. 39) trvá zřejmě ze všech velkých frameworků nejdéle.

Obr 39: Oficiální logo frameworku Symfony

8.6.1 Obecné vlastnosti

Trochu nezvyklou vlastností je vkládání funkcí přes příkazovou řádku, což může být problém spjatý s hostingem.

8.6.2 Ukázky kódu

Jelikož aplikace se píší za pomocí příkazové řádky, tak si uvedeme několik takových příkazů:

Příkaz pro vytvoření aplikace *frontend*:

```
$ php symfony generate:app frontend
```

Příkaz pro smazání veškeré cache:

```
$ php symfony cache:clear
```

Příkaz pro nápovědu jaké funkce má například příkaz *generate*:

```
$ php symfony list generate
```

8.6.3 Příklad webu vytvořeného pomocí Symfony

Tento framework má svůj vlastní ukázkový příklad, který se jmenuje Jobeet [55]. Tuto ukázku mohou uživatelé libovolně editovat a pracovat s ní, protože se každých 24 hodin automaticky sama přepíše do původního stavu.

8.6.4 Zhodnocení

Framework Symfony je jedním z nejsložitějších frameworků vůbec. Je velice obtížné jej pochopit, ovšem v momentě, kdy se jej uživatel naučí používat, dostává se mu do ruky mocný nástroj pro vytváření aplikací. Právě složitost frameworku způsobuje malý zájem. Většina uživatelů raději zvolí framework, který se dokážou rychle naučit.

8.7 Celkové zhodnocení frameworků

Popsali jsme si a ukázali příklady pěti různých PHP frameworků. Objektivně lze tvrdit, že nejpoužívanějším a nejrozšířenějším je Zend Framework [39]. Avšak vzhledem k tomu, že se český Nette Framework (NF) velice rychle rozvíjí, začal mu výrazně konkurovat. Důvodem jeho vzniku byly mimo jiné nedostatky jiných frameworků. O tom se zmínil také autor frameworku David Grudl [12] na prezentaci NF [17], konané v budově CPIT na VŠB. NF je více zaměřený na potřeby uživatelů, a i přes nedokončený Quick Start [15] nebylo těžké pochopit, jak se v tomto frameworku vytvářejí složitější aplikace. Velkou předností NF je podpora české komunity [13]. Framework Symfony [51] je velmi mocným nástrojem, ale je dosti složité pochopit ho a naučit se s ním pracovat. CakePHP [47] a CodeIgniter [43] jsou pak doporučovány spíše pro jednodušší aplikace.

Na základě provedeného srovnání preferuji Framework Nette. Při volbě frameworku je však vždy potřeba přihlédnout na požadavky zadání a složitost aplikace.

9. Přiložená aplikace

Přiložená aplikace bude umístěna na firemním serveru společnosti BKB Metal a.s. a na CD (Příloha A). Systém je plně dokončen podle specifikace požadavků firmy. Po domluvě s firmou BKB Metal a.s. bude diplomant sloužit jako správce aplikace, který do budoucna bude provádět aktualizace a úpravy aplikace, které již nejsou součástí diplomové práce. Společně s aplikací jsou dodány dvě příručky: programátorská příručka a uživatelská příručka.

V základní verzi aplikace jsou vytvořeny dva přístupy:

- Uživatel s rolí „User“: Login: user@test.cz Heslo: user
- Uživatel s rolí „Admin“: Login: admin@test.cz Heslo: admin

Přístupy jsou voleny tak, aby pokryly obě důležité role v aplikaci. Díky tomu mohou zaměstnanci společnosti BKB Metal a.s. vidět rozdíly mezi jednotlivými rolemi již před prostudováním příruček. Aplikace je plně zabezpečena proti útokům a díky tomu jsou data chráněna.

10. Závěr

Předkládaná práce popisuje vytvoření aplikace pro správu vozového parku společnosti BKB Metal a.s. V kapitole č. 3 Specifikace zadání je čtenář obeznámen s požadavky společnosti, uvedenými v zadání. Zadání bylo firmou přesně specifikováno, aby v budoucnu nedošlo k nedorozumění a následné nutnosti přepracovat aplikaci. Poté byla zpracována analýza a návrh celé aplikace, které jsou v kapitole č. 4 rozděleny do tří částí: Datová analýza, Funkční analýza a Časová analýza. V diplomové práci jsou uvedeny pouze ukázky z těchto jednotlivých částí. Všechny jsou pak umístěny v Programátorské příručce, která je součástí přílohy (Příloha A).

Práce popisuje krok po kroku implementaci celé aplikace. Vývoj byl proveden za pomocí PHP aplikačního rámce Nette. Bylo použito PHP verze 5.2.11. Databáze byla vytvořena za pomocí MySQL verze 5.1.36 s možností správy databázových tabulek standardním nástrojem phpMyAdmin. Po dokončení implementace byla zhodnocena funkčnost celé aplikace.

Součástí diplomové práce je stručný popis jazyka PHP a porovnání použitého PHP aplikačního rámce Nette s ostatními nejpoužívanějšími PHP aplikačními rámci. Výsledek porovnávání je vyhodnocen.

Zpětný pohled na celou diplomovou práci ukáže, že vytvoření aplikace správy vozového parku pro firmu BKB Metal a.s. nebylo jednoduché. Musely být dodrženy všechny prvky správného postupu pro vytváření takovéto aplikace a důkladně promyšlena všechna omezení funkcí jednotlivých rolí. Tato aplikace je mým druhým rozsáhlejším informačním systémem vytvořeným pomocí aplikačního rámce Nette. S tímto aplikačním rámcem jsem se seznámil již dříve a domnívám se, že jeho výběr byl i v tomto případě dobrým krokem.

Aplikace správy vozového parku pro firmu BKB Metal a.s. bude nasazena na lokálním serveru firmy, kde bude zahájen několikaměsíční zkušební provoz. Během této doby mohou ze strany zaměstnanců vzniknout požadavky na přizpůsobení či poupravení aplikace. Tyto požadavky budou následně konzultovány s vedením firmy a v rámci možností bude systém aktualizován.

Na základě zkušeností získaných při vývoji informačních systémů se domnívám, že by bylo vhodné, aby aplikace obsahovala ještě jednu uživatelskou roli, která by nebyla zahrnuta v divizích. Uživatel s touto rolí by pak mohl provádět zcela výjimečné operace, jako je mazání divizí a zaměstnanců. Tyto funkce v systému nebyly na požadavek zadavatele implementovány, avšak v určitých situacích by mohly být užitečné.

Během vývoje této aplikace proběhlo několik konzultací se zadavatelem v prostorách firmy BKB Metal a.s., na kterých byly probírány dotazy a návrhy upřesňující vývoj aplikace. Ty byly následně zpracovány a implementovány do aplikace. Na základě kladných ohlasů zadavatele projevených během jednotlivých konzultací lze říci, že aplikace správy vozového parku splnila očekávání.

Literatura

[1] ŠARMANOVÁ, Jana. *Databázové a informační systémy: Učební text* [online]. Ostrava: VŠB-TU, 2007 [cit. 2010-04-21]. Dostupné z WWW: <http://www.elearn.vsb.cz/archivcd/FEI/DAIS/DAIS.pdf>. ISBN 978-80-248-1499-5.

[2] RATSCHILLER, Tobias. *phpMyAdmin* [online]. 1998, 2010 [cit. 2010-04-21]. Dostupné z WWW: <http://www.phpmyadmin.net/>.

[3] FIALA, Martin. *ABC Linuxu* [online]. 2004-08-21, 2009-03-21 [cit. 2010-05-02]. Hash. Dostupné z WWW: <http://www.abclinuxu.cz/slovnik/hash>.

[4] JAMES, Mark. *FAMFAMFAM* [online]. Verze 1.3. 2006 [cit. 2010-04-23]. Silk Icons. Dostupné z WWW: <http://www.famfamfam.com/lab/icons/silk/>.

[5] GRAMES, Martin. *Chapadlo* [online]. 2007 [cit. 2010-04-25]. Hackerem sám sobě (1) - PHP Injection. Dostupné z WWW: <http://www.chapadlo.cz/weblog/clanek/hackerem-sam-sobe-1-php-injection>.

[6] DOČEKAL, Daniel. *Lupa* [online]. 2008-06-10 [cit. 2010-04-25]. Na podceňovanou hrozbu SQL injection doplácí i řada českých webů. Dostupné z WWW: <http://www.lupa.cz/clanky/na-sql-injection-doplaci-i-rada-ceskych-webu/>.

[7] TICHÝ, Jan. *PHP Guru* [online]. 2008-02-22 [cit. 2010-05-03]. Cross-site scripting. Dostupné z WWW: <http://www.phpguru.cz/clanky/cross-site-scripting>.

[8] VRÁNA, Jakub. *PHP triky* [online]. 2006-04-24 [cit. 2010-04-25]. Cross-Site Request Forgery. Dostupné z WWW: <http://php.vrana.cz/cross-site-request-forgery.php>.

[9] BRÁZA, Jiří. *PHP 5 : začínáme programovat*. Praha: Grada, 2005. 244 s. Dostupné z WWW: <http://www.worldcat.org/title/php-5-zaciname-programovat/oclc/85141097>. ISBN 80-247-1146-X.

[10] WALES, Jimmy; SANGER, Larry. *Wikipedie : Otevřená enciklopedie* [online]. 2001-01-15 [cit. 2010-04-27]. Dostupné z WWW: <http://cs.wikipedia.org/>.

[11] Nette Foundation. *Nette Framework* [online]. 2008-12-11, 2010-05-02 [cit. 2010-05-03]. Dostupné z WWW: <http://nettephp.com>.

[12] GRUDL, David. *David Grudl.com* [online]. 2000 [cit. 2010-05-03]. Dostupné z WWW: <http://davidgrudl.com/>.

[13] Nette Foundation. *Nette Framework* [online]. 2008-12-11, 2010-01-22 [cit. 2010-05-03]. Kominuta. Dostupné z WWW: <http://nettephp.com/cs/komunita>.

[14] Nette Foundation. *Nette Framework forum* [online]. 2008-12-11 [cit. 2010-05-03]. Dostupné z WWW: <http://forum.nettephp.com/cs/>.

[15] GRUDL, David, et al. *Nette Framework* [online]. 2009-11-04, 2010-04-25 [cit. 2010-05-03]. Quick Start. Dostupné z WWW: <http://doc.nettephp.com/cs/quickstart>.

[16] *Institut Školení PHP* [online]. 2008 [cit. 2010-05-03]. Vývoj webových aplikací v Nette Framework. Dostupné z WWW: <http://www.skoleniphp.cz/skoleni-nette-vyvoj-webovych-aplikaci>.

[17] *Nette Framework forum* [online]. 2009 [cit. 2010-05-03]. Vývoj webových aplikací v PHP a Nette Framework. Dostupné z WWW: <http://forum.nettephp.com/cs/2840-david-o-nette-v-ostrave>.

[18] GRUDL, David. *PhpFashion* [online]. 2008, 2010-04-24 [cit. 2010-05-03]. Poslední sobota – sraz Nette Framework. Dostupné z WWW: <http://phpfashion.com/posledni-sobota-sraz-nette-framework>.

[19] DANĚK, Petr. *Root.cz* [online]. 2008-09-11 [cit. 2010-05-03]. Velký test PHP frameworků: Zend, Nette, PHP a RoR. Dostupné z WWW: <http://www.root.cz/clanky/velky-test-php-frameworku-zend-nette-php-a-ror>.

[20] GRUDL, David. *Nette Framework* [online]. 2007-06-25, 2010-04-04 [cit. 2010-05-03]. Licenční politika. Dostupné z WWW: <http://nettephp.com/cs/licence>.

[21] GRUDL, David. *Nette Framework* [online]. 2008-12-17, 2010-05-02 [cit. 2010-05-03]. Doplňky, pluginy a komponenty. Dostupné z WWW: <http://addons.nettephp.com/cs/>.

[22] FOWLER, Martin. *Patterns of Enterprise Application Architecture*. USA : Pearson Education, Inc., 2002. Web Presentation Patterns, s. 560. ISBN 0-321-12742-0.

[23] CAULDWELL, Patrick. *Code Leader : Using People, Tools, and Processes to Build Successful Software*. USA : Wiley Publishing, Inc., 2008. The Model-View-Presenter (MVP) Model, s. 233. ISBN 978-0-470-25924-5.

[24] AMBLER, Scott. *Agile Database Techniques : Effective Strategies for the Agile Software Developer*. USA : Wiley Publishing, Inc., 2003. Mapping Objects to Relational Databases, s. 480. ISBN 0-471-20283-5.

[25] Nette Foundation. *Dibi : tiny 'n' smart database layer* [online]. 2008, 2010 [cit. 2010-05-03]. Dostupné z WWW: <http://dibiphp.com/cs/>.

[26] MAREK, Jan. *Nette Framework* [online]. 2010 [cit. 2010-05-03]. Ormion. Dostupné z WWW: <http://addons.nettephp.com/cs/ormion>.

[27] MAREK, Jan. *Jan Marek : portfolium webdesignera* [online]. [cit. 2010-05-03]. Dostupné z WWW: <http://www.janmarek.net/>.

[28] SMIČKA, Radim. *Optimalizace pro vyhledávače – SEO: Jak zvýšit návštěvnost webu* [online]. Dubany: Jaroslava Smičková, 2004 [cit. 2010-05-03]. Dostupné z WWW: <http://seo.jasminka.cz/seo-kniha.pdf>. ISBN 80-239-2961-5.

[29] *PHP* [online]. [cit. 2010-05-03]. Htmlspecialchars. Dostupné z WWW: <http://cz.php.net/htmlspecialchars>.

[30] Nette Foundation. *Nette Framework* [online]. 2009 [cit. 2010-05-04]. Dokonalé zabezpečení webových aplikací. Dostupné z WWW: <http://nettephp.com/cs/hlavni-prednosti/bezpecnost>.

[31] New Digital Group, Inc. *Smarty: Template Engine* [online]. 2002 [cit. 2010-05-03]. Dostupné z WWW: <http://www.smarty.net/>.

[32] FLORY, Ian. *Doplňky aplikace Firefox* [online]. 2006, 2009-07-09 [cit. 2010-05-03]. DBGbar. Dostupné z WWW: <https://addons.mozilla.org/cs/firefox/addon/3227>

[33] *Webový prohlížeč Firefox* [online]. [cit. 2010-05-03]. Dostupné z WWW: <http://www.mozilla.com/cs/>.

[34] HEWITT, Joe, et al. *Doplňky aplikace Firefox* [online]. 2006, 2009-07-09 [cit. 2010-03-12]. Firebug. Dostupné z WWW: <https://addons.mozilla.org/cs/firefox/addon/1843>.

[35] DORN, Christoph. *Doplňky aplikace Firefox* [online]. 2006, 2009-12-05 [cit. 2010-05-03]. FirePHP. Dostupné z WWW: <https://addons.mozilla.org/cs/firefox/addon/6149>.

[36] Oracle Corporation. *NetBeans* [online]. 2010 [cit. 2010-05-03]. Dostupné z WWW: <http://netbeans.org/>.

[37] The Eclipse Foundation. *Eclipse* [online]. 2010 [cit. 2010-05-03]. Dostupné z WWW: <http://www.eclipse.org/>.

[38] *Václav Klaus* [online]. 2010 [cit. 2010-05-03]. Dostupné z WWW: <http://www.klaus.cz/>.

[39] *Root.cz* [online]. 1998 [cit. 2010-05-03]. Dostupné z WWW: <http://www.root.cz/>.

[40] Nette Foundation. *Nette Framework* [online]. 2008 [cit. 2010-05-04]. Nette\Forms. Dostupné z WWW: <http://doc.nettephp.com/cs/nette-forms>.

[41] Zend Technologies Ltd. *Zend Framework* [online]. 2006 [cit. 2010-05-03]. Dostupné z WWW: <http://framework.zend.com/>.

[42] Zend Technologies Ltd. *Zend Studio: The Professional PHP IDE* [online]. 2009 [cit. 2010-05-03]. Dostupné z WWW: <http://www.zend.com/en/products/studio/>.

[43] MITON CZ, s.r.o. *Výsledky.cz: Sportovní výsledky na dlani* [online]. 2006 [cit. 2010-05-03]. Dostupné z WWW: <http://www.vysledky.cz/>.

[44] W3W. *Shopio : Internetový obchod* [online]. 2008 [cit. 2010-05-03]. Dostupné z WWW: <http://www.shopio.cz/>.

[45] EllisLab, Inc. *CodeIgniter: Open source PHP web aplication framework* [online]. 2001, 2010 [cit. 2010-05-03]. Dostupné z WWW: <http://codeigniter.com/>.

[46] EllisLab, Inc. *EllisLab: Where Ideas Heatch!* [online]. 2002 [cit. 2010-05-03]. Dostupné z WWW: <http://ellislab.com/>.

[47] Pacific Peoples' Partnership. *Pacific Peoples' Partnership: Climate Change* [online]. 2008 [cit. 2010-05-03]. Dostupné z WWW: <http://climatechange.pacificpeoplespartnership.org/>.

[48] *Nuherbs co: The Future of Chinese Medicine* [online]. 2010 [cit. 2010-05-03]. Dostupné z WWW: <http://nuherbs.com/>.

[49] Cake Software Foundation, Inc. *CakePHP: the rapid development php framework* [online]. 2005 [cit. 2010-05-03]. Dostupné z WWW: <http://cakephp.org/>.

[50] HANSSON, David Heinemeier. *Ruby on Rails: Web development that doesn't hurts* [online]. 2003 [cit. 2010-05-03]. Dostupné z WWW: <http://rubyonrails.org/>.

[51] iNET Interactive. *HotScripts: The net's largest PHP, CGI, Perl, JavaScript and ASP script collection and resource web portal* [online]. 2010 [cit. 2010-05-03]. Dostupné z WWW: <http://www.hotscripts.com/>.

[52] STORY, Mark. *Mark Strory* [online]. 2003 [cit. 2010-05-03]. Dostupné z WWW:<http://mark-story.com/>.

[53] Sensio Labs. *Symfony : Web PHP Framework* [online]. 2005 [cit. 2010-05-03]. Dostupné z WWW: <http://www.symfony-project.org/>.

[54] EVANS, Clark. *The Official YAML Web Site* [online]. 2001 [cit. 2010-05-04]. Dostupné z WWW: <http://www.yaml.org/>.

[55] Sensio Labs. *Jobeet : Your best job board* [online]. [cit. 2010-05-03]. Dostupné z WWW: <http://www.jobeet.org/>.

Příloha A

Obsah CD

Na přiloženém CD naleznete adresáře:

- **Aplikace**
  - adresář obsahující všechny soubory aplikace pro správu vozového parku
- **Dokumenty**
  - adresář obsahující předložený text, Programátorská příručka a Uživatelská příručka

V adresáři „Aplikace“ se nacházejí dva soubory „readme.txt“. V jednom je popsána adresářová struktura aplikace a ve druhém je uveden jeden z možných způsobů instalace software, který je potřebný pro spuštění aplikace:

- **Popis struktury aplikace**
  - Nachází se v souboru „CD:\Aplikace\bkbmetal.zip\bkbmetal\app\readme.txt”
- **Popis instalace potřebného software**
  - Nachází se v souboru „CD:\Aplikace\bkbmetal.zip\bkbmetal\readme.txt”

V adresáři „Dokumenty“ jsou všechny soubory uloženy ve formátu PDF a Microsoft Office 2007:

- **Předkládaný text** (celkem 78 stran včetně vloženého zadání)
  - DP_KRH013.pdf
  - DP_KRH013.docx
- **Programátorská příručka** (celkem 52 stran)
  - PP_KRH013.pdf
  - PP_KRH013.docx
- **Uživatelská příručka** (celkem 35 stran)
  - UP_KRH013.pdf
  - UP_KRH013.docx